Шифр **«Гібрид-Ліс»**

Гібридна система класифікації типів уражень легень при Covid-19 на основі нейронної мережі та самоорганізованого лісу

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

пкс – піксель

МГУА – метод групового урахування аргументів

GLCM – Gray-Level Co-occurrence Matrix

GLRLM – Gray Level Run Length Matrix

GLSZM – Gray Level Size Zone Matrix

GLDM – Gray Level Dependence Matrix

NGTDM – Neighbouring Gray Tone Difference Matrix

ROI – region of interest (область інтересу)

ВСТУП

Пандемія COVID-19 стала однією з найяскравіших глобальних реакцій природи на безперервне зростання людської популяції. Тому ефективні діагностичні та аналітичні інструменти для боротьби з наслідками інфекції сьогодні дійсно потрібні. Проблема в тому, що, за даними ВООЗ, навіть найпопулярніші ПЛР-тести не можуть дати достовірних результатів і їх точність становить близько 70%. У той же час комп'ютерна томографія дозволяє виявити ураження легень при Covid-19 з набагато вищим ступенем достовірності – до 99%. При цьому оцінка стану хворого в значній мірі визначається кількісними показниками різного типу ураження легень (матове скло, бруківка, консолідація). В роботі розглядається застосування гібридної системи класифікації на основі згорткової нейронної мережі та удосконаленого алгоритму самоорганізованого лісу для вирішення задачі класифікації типів уражень легень при Covid-19 на знімках комп'ютерної томографії.

Мета роботи: Підвищення точності систем класифікації типів уражень легень при Covid-19 на знімках комп'ютерної томографії.

Основні задачі :

- Розробка алгоритму та програмної реалізації для формування класифікатору на основі удосконаленого алгоритму самоорганізованого лісу.
- Розробка гібридної системи класифікації типів уражень легень при Covid-19 на основі згорткової нейронної мережі та самоорганізованого лісу

РОЗДІЛ 1
АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1.1 Постановка задачі класифікації типів уражень легень при захворюванні Covid-19

Передбачається, що існує скінченна кількість класів D^*_i, $i = 1, ..., K$, K=3, уражень легеневої тканини пацієнтів з захворюванням на COVID-19 що представляють собою наступні типи:

1. «ground-glass opacity» (рис. 1.1)
2. «crazy-paving» (рис. 1.2)
3. «consolidation» (рис. 1.3)

Класи відображаються у вигляді наборів зображень (об’єктів класифікації d*), які представлені у вигляді областей інтересу (ROI) КТ-зображень легенів пацієнтів. Кожен клас D^*_i, $i = 1, ..., K$ є скінченним або нескінченним набором зображень ROI d^*. Окрім того, передбачається (1.1).

$$D^*_i \cap D^*_j = \emptyset,\ i \neq j \qquad (1.1)$$

Класи надані нам скінченними навчальними підмножинами D_i потужності n_i, $i = 1,..., K$, представлені об’єктами $(ROI)_{ij}$, де $j = 1,..., n_i$. Кожен об’єкт d_{ij} є фрагментом КТ-зображення легенів людини $(ROI)_{ij}$, який позначений як патологічний. На основі наведених навчальних підмножин D_i, $i = 1, ..., K$ необхідно

створити механізм найкращої класифікації об'єктів d^*_{ij} з D_i, $i = 1, ..., K$ в обраному класі.

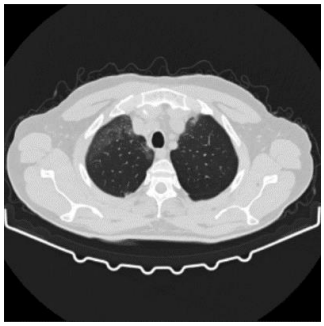

Рис. 1.1 – «ground-glass opacity»

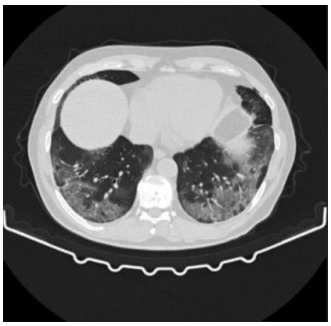

Рис. 1.2 – «crazy-paving»

Рис. 1.3 – «consolidation»

1.2 Аналіз літературних джерел

Існує два домінуючих підходи до аналізу медичного зображення з метою класифікації патологій. Перший базується на аналізі вхідних зображень, тобто з використанням інтенсивності пікселів зображення як вхідних ознак. В роботах [1, 2] при класифікації типів ураження легень при COVID-19 значення інтенсивності пікселів використовуються, як вхідні дані для згорткової нейронної мережі (CNN). У [3] CNN використовуються для вилучення ознак із інтенсивностей сполучень пікселів зображення. Другий підхід базується на аналізі характеристик текстури області інтересу (ROI). Різні статистичні характеристики текстури, розраховані в [4, 5] використовуються, як вхідні ознаки для класифікатора. У роботі [6] пропонується аналізувати, як інтенсивність пікселів, так і статистичні характеристики текстури. Було виявлено деякі проблеми, пов'язані з класифікацією різних класів ROI уражень легень, виконаною безпосередньо за інтенсивністю пікселів у [1, 2]. Це пов'язано з прогресуючим характером ураження при COVID-19, "crazypaving" є наслідком прогресування патології "ground-glass", а "консолідація" - наслідком прогресування "crazypaving", У прикордонних областях діапазон

одиниць Хаунсфілда в ROI приблизно однаковий, але існують помітні візуальні відмінності текстур в областях інтересу різних типів. Даний факт обгрунтовує при класифікації типів легеневих уражень легень при COVID-19 використання самє текстурних характеристик, які відображають характеристики залежності значень інтенсивності сусідніх пікселів зображення.

Висновки до розділу 1

Наведено постановку задачі класифікації типів ураження легень при захворювання на Covid-19. Для поставленої задачі роботи проаналізовано літературні джерела та методи вирішення споріднених задач, обгрунтовано застосування тектурних характеристик.

РОЗДІЛ 2
АЛГОРИТМ МОДЕЛЮВАННЯ ДЕРЕВ САМООРГАНІЗОВАНОГО ЛІСУ ЗА КРИТЕРІЄМ ЯКОСТІ ПРОГНОЗУ КЛАСИФІКАЦІЇ

2.1 Постановка задачі

Існує ряд високоефективних алгоритмів класифікації з класу методу Random Forest. Всі вони засновані на створені спеціалізованих (з точки зору підмножин об'єктів та ознак) класифікаторів-дерев та формуванні результату класифікації на основі колективного прийняття рішення. У роботі ставиться завдання удосконалення алгоритмів класу Random Forest щоб підвищити ефективність класифікації за рахунок застосування принципів самоорганізації за критерієм якості прогнозу класифікації моделей-дерев та удосконалення функції голосування дерев лісу за рахунок застосування логістичного перетворення.

2.2 Методи та інструменти

2.2.1 Випадковий ліс прийняття рішень

Вперше алгоритм випадкового лісу був запропонований китайським вченим на ім'я Тін Кам Хо у 1995 році [7]. Цей алгоритм є представником класу метаалгоритмів ансамблевого навчання для класифікації, регресії та вирішення інших задач, в основі якого лежить одночасна побудова скінченної множини незалежних дерев прийняття рішень. Випадковий ліс з допомогою детермінованої функції усереднення поєднує рішення ансамблю моделей, що дозволяє отримати результати прогнозування з більшою точністю.

Алгоритми випадкового лісу на сьогоднішній день залишаються одним з найпопулярніших рішень сучасних задач та широко застосовується у задачах

класифікації різної складності, що підтверджує актуальність такого ансамблевого алгоритму[8].

Випадковий ліс — популярний алгоритм машинного навчання, який належить до класу навчання з учителем. Його можна використовувати, як для задач класифікації так і для завдань пошуку регресій. Він заснований на концепції ансамблевого навчання, що являє собою процес об'єднання кількох класифікаторів для вирішення складної проблеми та покращення продуктивності моделі [7].

Як випливає з назви, випадковий ліс — це класифікатор, який містить ряд дерев рішень для різних підмножин даного набору даних та ознак і усереднює результати, щоб підвищити точність прогнозування цього набору даних. Замість того, щоб покладатися на одне дерево рішень, випадковий ліс бере прогноз з кожного дерева на основі більшості голосів передбачень і прогнозує кінцевий результат.

Структура лісу (кількість дерев у лісі), визначається за критерієм точності на валідаційній вибірці, що запобігає проблемі перенавчання лісу.

Діаграма нижче пояснює роботу алгоритму (рис. 2.1):

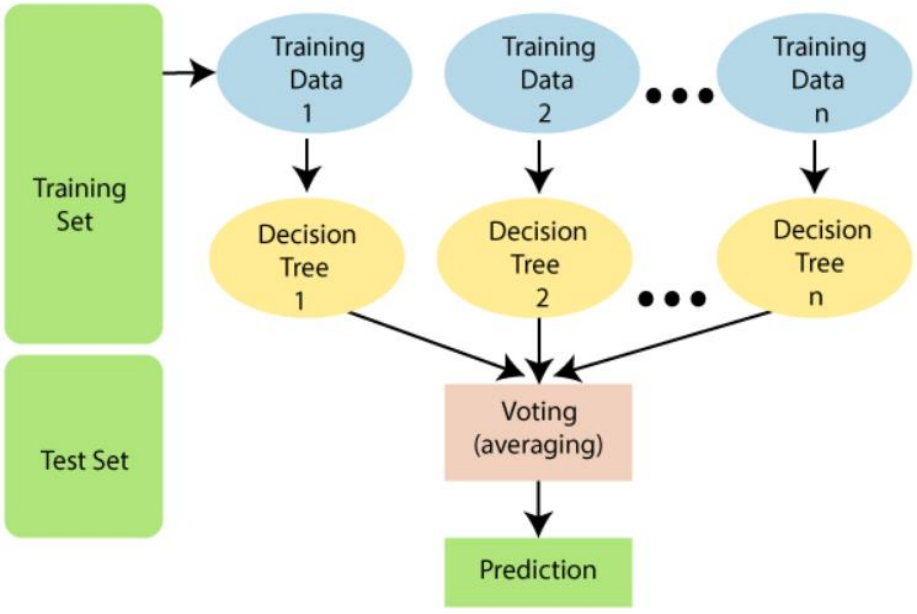

Рисунок 2.1 – Діаграма алгоритму випадкового лісу

Алгоритм випадкового лісу складається з двох фаз:

1. Створення випадкового лісу шляхом об’єднання N дерева рішень
2. Створення прогнозів для кожного дерева, створеного на першому етапі.

Принципи створення лісу у класичній версі алгоритму можна пояснити на наступних кроках:

Крок 1. Обрати K випадкових точок даних із навчального набору.

Крок 2. Побудувати дерево прийняття рішень на підмножині даних, що була обранана попередньому кроці. Вузли дерева будуються з врахуванням певної підмножини ознак на кожному кроці.

Крок 3: Повторювати кроки 1 і 2 допоки буде отримано N дерев, де N визначається мінімумом помилки класифікації на валідаційній вибірці даних

Крок 4. Для точок даних із тестового набору необхідно отримати прогнози кожного дерева рішень і класифікувати кожну точку до тої категорії, для якої кількість прогнозів серед дерев виявилось найбільше.

2.2.2 МГУА

Метод групового урахування аргументів (МГУА) – метод, що належить до категорії підходів індуктивного моделювання [9]. Область його застосування – відносно невеликі навчальні вибірки даних, метод здатний об’єктивно оптимізувати структуру моделей. Принципи МГУА лежать в основі найбільш застосовуємих ефективних засобів моделювання – генетичних алгоритмів, методів індуктивного моделювання, глибинного навчання.

В загальному випадку, зв'язок між вхідними та вихідними змінними моделі можна апроксимувати функціональним рядом Вольтерра, дискретним аналогом якого є поліном Колмогорова-Габора (2.2):

$$y = a_0 + \sum_{i=1}^{m} a_i x_i + \sum_{i=1}^{m}\sum_{j=1}^{m} a_{ij} x_i x_j + \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m} a_{ijk} x_i x_j x_k + \cdots \qquad (2.1)$$

де $x=(x_1, x_2, \ldots, x_m)$ – вектор вхідних змінних, $A=(a_0, a_1, a_2, \ldots, a_m)$ – вектор вагів. Поліном Колмогорова-Габора може апроксимувати будь-яку стаціонарну випадкову послідовність спостережень і може бути обчислений або адаптивними методами, або системою нормального рівняння Гаусса.

Автор [12], запропонував нову групу алгоритмічних процедур, що було названо «Методом групового урахування аргументів (МГУА)». Започатковано метод було з багаторядного алгоритму, де дотримуючись підходу персептронного типу, автор показав що можливо апроксимувати поліном Колмогорова-Габора, використовуючи поєтапно поліноми низького порядку, наприклад (2.2) для кожної пари вхідних змінних.

$$y = a_0 + a_1 x_i + a_2 x_j + a_3 x_i x_j + a_4 {x_i}^2 + a_5 {x_j}^2 \qquad (2.2)$$

Під час процедури моделювання алгоритм МГУА включає чотири евристики, які представляють основні особливості теорії методу [11]:

1. Накопичення набору спостережень, який потенційно може має відношення до досліджуваного об’єкта.
2. Розділення набору даних на дві групи. Перша буде використовуватися для оцінки коефіцієнтів (вагів) моделі, а другий виділить корисну інформацію, що прихована у вхідних даних.
3. Створення набору елементарних функцій, складність яких буде зростати в результаті ітераційної процедури, що створюватиме більш комплексні моделі.
4. Застосування зовнішнього критерію для вибору оптимальної моделі.

Автори дослідження [5] запропонували шляхи удосконалення в класі алгоритмів Random Forest з використанням приципів методів групового урахування аргументів. Таким чином, кожне окреме дерево за результатами навчання мало більшу точність класифікації, що призвело до покращення результатів прогнозування ансамблевого алгоритму. В даній роботі продовжено удосконалення

алгоритму самоорганізації дерев на принципах МГУА. Застосовано новий алгоритм застосування зовнішнішнього критерію для вибору структури дерев, що здійснює вибір кращої ознаки у вузел на поточному рівні дерева, що забезпечує найкращий прогноз класифікації на (заданому) к-тому рівні дерева [10].

2.2.3 Логістична регресія

Модель логістичної регресії приймає натуральний логарифм шансів як функцію регресії від предикторів [13]. Використовується у випадку, коли залежна змінна є бінарною. Із використанням значення порогу може застосовуватись як метод класифікації.

У даній роботі автори дослідження [10] застосували логістичну регресію як функцію агрегації прогнозів самоорганізованих моделей. За результатами дослідження, були отримані результати підвищеної точності у порівнянні із класичним алгоритмом випадкового лісу.

2.3 Агоритм побудови дерев самоорганізованого лісу за критерієм якості прогнозу класифікації

2.3.1 Алгоритм лісу

Кожне дерево лісу представляє собою бінарне дерево, у вузлах якого обчислюються ознаки із відповідними порогами та знаками. Класифікація здійснюється шляхом порівняння значення ознаки об'єкта з пороговим значенням та знаком. Якщо значення ознаки X_i більше порогу P_i зі знаком «>», то шлях об’єкта, що класифікується пролягає через правого нащадка, інакше – через лівого. У випадку знака «<» вибір шляху інвертується. Процедура виконується до тих пір, поки об’єкт не досягне листового вузла, який і визначатиме клас об’єкта. Саме цей листовий вузол стане вирішальним у прогнозі. Далі йтиме алгоритм побудови дерева:

Попередній етап. В процесі передобробки даних збільшується простір об’єктів від X до f та розбивається вибірка.

Набір даних R розбивається на множини A, B, C. Тут нехай $n_{AB} = 0,8 * n_R$, C = $0,2 * n_R$. Кожне дерево будується на наборі AB, згідно з (2.3):

$$n_A = (0,7 \pm k_i)n_{AB}, n_B =(0,3 \mp k_i)n_{AB} \quad (2.3)$$

тут k_i – випадкове число від [0, - 0,05]. Співвідношення (2.3) забезпечує мінливість ознак і порогів у вузлах при побудові і-го дерева лісу. Набір даних розподілений стратифікованим способом рівномірно по дисперсії відносно центрів класів у просторі ознак f.

Побудова вузла. Пошук кращого порога (рис. 2.4) для ознаки *i* з *f* у вузлі відбувається за F^{sc}-метрикою на A (далі F^{scA}) множини P_i (2.4) за виключенням діапазонів, де значення ознак на варіаційному ряді об’єктів за номером класу слідують підряд.

$$\{p_i \in P_i \mid p_i \in \{\frac{x_{j+1}+x_j}{2}, j = 1,..,n_i, x_j \in [c_{0i}; c_{1i}]\}\} \quad (2.4)$$

де p_i – значення порогу для ознаки i, x_j – j-те значення ознаки із множини порогів, n_i – кількість елементів множини P_i, c_{0i}, c_{1i} – центри класів за ознакою i.

Вибір F кращих ознак f_{best} для вузла відбувається за комбінованою F-метрикою на A + B відповідно (2.5).

$$F^{sc*} = w \cdot F^{scB} + (w - 1) \cdot F^{scA} \quad (2.5)$$

Для кожної ознаки з f_{best} будуються лівий та правий нащадки. Оскільки нащадки класифікують точки даних на підмножинах, що не перетинаються, це дозволяє скористатись наступним фактом: F^*-метрика суми нащадків дорівнює сумі F^*-метрик. Кращою вважається ознака, для якої максимізується (2.6)

$$F^{sc*}_{total} = F^{sc*}_{left} + F^{sc*}_{rigth} \quad (2.6)$$

, де F^{sc*}_{left} – F^*-метрика лівого нащадка, F^{sc*}_{rigth} – F^*-метрика правого нащадка. Краща ознака зберігається в корені разом із порогом та знаком.

Таким чином, краща ознака на поточному рівні p обирається не тільки за власною , а й за найкращими результатами на рівні p+1, що дозволяє отримати кращі результати в цілому.

Побудова усього дерева починається з вузла-кореню та проводиться в ширину та в висоту. Критерій зупинки у розростанні дерева визначається максимальною глибиною дерева *d*, або ж за умови, що F^{sc*} усього дерева після побудови поточного вузла не покращилась. У разі досягнення критерію зупинки, поточний вузол залишає за собою ознаку із порогом, що найкраще класифікує дані. Кожне дерево після завершення навчання класифікує вхідні дані із множини AB та повертає вектор прогнозів. Останній складає множину ознак для логістичної регресії, де кожна ознака індексується порядковим номером дерева (рис. 2.2).

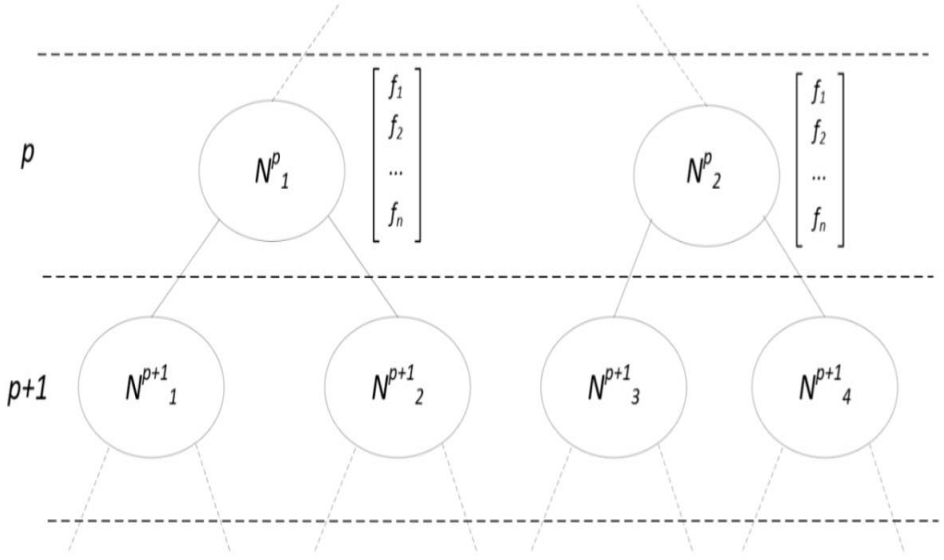

Рисунок 2.2 – Два послідовних рівня дерева в самоорганізаційному лісі

Навчання логістичної регресії відбувається на множині AB, а тестування – на екзаменаційній множині C. Кількість дерев в лісі збільшується поки метрика якості логістичної регресії на вибірці B зростає.

2.3.2 Застосування вибору елементів структури поточного m-того рівня дерев за зовнішнім критерієм прогнозу класифікації на (m+q)-тому рівні

У підрозділі 2.3.1 розглядається алгоритм для формування самоорганізованих дерев. У ньому при побудові вузла дерева виконується пошук найкращої ознаки разом із порогом та знаком, яка гарантуватиме найкращу продуктивність в нащадках вузла, оскільки вирішується задача максимізації (2.6).

Для покращення прогнозів моделі самоорганізованого дерева, пропонується в алгоритмі узагальнити число q, що позначає кількість рівнів дерева в глибину, де відбувається порівняння F^{sc*}-метрик для пранащадків.

Таким чином формула (2.6) приймає новий вигляд (2.7)

$$F^{sc*}{}_{total} = \begin{cases} F^{sc*}_{left} + F^{sc*}_{right}, & q = 1 \\ F^{sc*}_{total,left} + F^{sc*}_{total,right}, & q \geq 2 \end{cases} \qquad (2.7)$$

, де $F^{sc*}{}_{total,left}$, $F^{sc*}{}_{total,right} - F^{sc*}{}_{total}$ для лівого та правого нащадків відповідно.

Покращений алгоритм потенційно може мати кращі результати, оскільки в процесі побудови кожного вузла дерева відбувається пошук такої умови, що дасть найкращі результати на q рівнів уперед, що призведе до покращення результатів прогнозування усього дерева.

Висновки до розділу 2

Розглянуто теоретичні засади методу самоорганізації МГУА, та області застосування логістичної регресії. Наведено основні теоретичні положення щодо алгоритму випадкового лісу.

Запропоновано удосконалення алгоритму касифікації «Випадковий ліс» за рахунок впроваждення принципів самоорганізації дерев за критерієм якості прогнозу класифікації та логістичного перетворення функції голосування, структура якої визначена за МГУА. Лістинг алгоритму наведений у додатку В.

РОЗДІЛ 3
ЗАСТОСУВАННЯ ГІБРИДНОГО КЛАСИФІКАТОРУ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ ТА САМООРГАДНІЗОВАНОГО ЛІСУ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ КЛАСИФІКАЦІЇ УРАЖЕНЬ ЛЕГЕНЬ

3.1 Постановка задачі

У цьому розділі пропонується вирішення прикладної задачі, сформульованої у підрозділі 1.1.

3.2 Побудова матриць текстурних характеристик

Для ряду завдань класифікації патологій по медичним зображенням найбільш ефективним є застосування текстурних характеристик. Прикладами таких завдань є класифікація патологій паренхіматозних органів (серце, печінка легені). Коротко наведемо основні засади визначення текстурних характеристик.

По своїй природі текстурні характеристики зображення є статистичними показниками зустрічальності поєднань кольорів пікселів у ньому. Статистичним методом дослідження текстури, який враховує просторове співвідношення пікселів, є матриця спільного виникнення сірого рівня (Gray-Level Co-occurrence Matrix – GLCM), також відома як матриця просторової залежності рівня сірого. Функції GLCM характеризують текстуру зображення та несуть інформацію про те, як часто в зображенні зустрічаються пари пікселів із певними значеннями та в конкретній просторовій орієнтації.

Побудова матриці GLCM [14] відбувається за алгоритмом, що описано нижче.

Для прикладу буде розглянуто синтетичне дрібне зображення (рис. 3.1). Значення на рис. 3.2 є рівнями сірого зображення. Чим нижче число рівня сірого, тим темніше зображення.

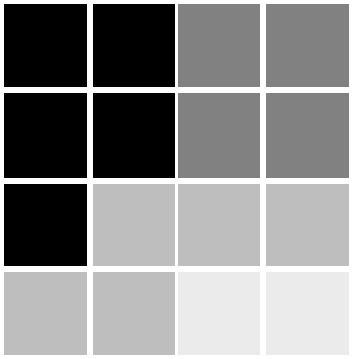

Рисунок 3.1 – тестове зображення 4x4 пкс для побудови матриці GLCM

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	3

Рисунок 3.2 – матриця рівнів сірого для тестового зображення на рис. 3.1.

Текстура GLCM розглядає відношення між двома пікселями одночасно, що мають назви опорного і сусіднього пікселя. За замовчуванням, сусідній піксель обирається праворуч від опорного. Кожен піксель на зображенні по черзі стає опорним пікселем. Пікселі вздовж правого краю не мають сусіда справа, тому вони не використовуються для побудови матриці в якості опорних пікселів. На рис. 3.3 показано кілька таких попарних зв’язків: червоний пікселів є опорним пікселем, а синій – сусіднім.

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	3

Рисунок 3.3 – демонстрація опорних та сусідніх значень матриці рівнів сірого

Основна ідея матриці GLCM полягає в підрахунку частоти появлення пар опорний-сусідній пікселі з однаковими відповідними значенням. Рядки матриці представляють собою значення, що можуть приймати опорні пікселі, а стовпчики – сусідні пікселі. Пара, що має опорний піксель з рівнем сірого *i*, та сусідній піксель з рівнем *j*, додаватиме одиницю у відповідну клітинку (*i*, *j*) матриці GLCM. Після обчислення кожної пари, отримаємо матрицю GLCM (рис. 3.4) для розглянутого зображення.

2	2	1	0
0	2	0	0
0	0	3	1
0	0	0	1

Рисуноок 3.4 – матриця GLCM для зображення на рис. 2.2.

Розглянута матриця була побудована для взаємного просторового відношення пікселів (1,0), тобто сусідній піксель знаходився строго праворуч на відстані одного пікселя. Проте можна застосовувати й інші просторові відношення, наприклад (-1,0) – сусідній піксель знаходитиметься ліворуч від опорного, (1,1) – по діагоналі справа знизу, (2,0) – відстань між пікселями дорівнюватиме 2 пкс. Для кожної конфігурації просторового відношення створюватиметься нова матриця GLCM.

Окрім того, існує ще декілька статистичних методів дослідження текстури, серед яких:

- GLRLM (Gray Level Run Length Matrix) – Матриця, що кількісно визначає довжину пробігу рівня сірого, яка визначається як довжина в кількості послідовних пікселів, що мають однакове значення рівня сірого.
- GLSZM (Gray Level Size Zone Matrix) – Матриця, що кількісно визначає зони рівня сірого на зображенні. Зона рівня сірого визначається як кількість з'єднаних пікселів, які мають однакову інтенсивність рівня сірого.
- GLDM (Gray Level Dependence Matrix) – Матриця, що кількісно описує залежність рівня сірого у зображенні. Залежність рівня сірого визначається як кількість з'єднаних пікселів на заданій відстані, які залежать від центрального пікселя.
- NGTDM (Neighbouring Gray Tone Difference Matrix) - Матриця, що кількісно визначає різницю між значенням сірого та середнім значенням сірого для його сусідів на заданій відстані.

3.3 Дані для задачі класифікації

Анонімні дані для розробки класифікатора надано ДУ «Національний інститут фтизіатрії та пульмонології імені Ф.Г. Яновського НАМН України». Набір даних складається з 1831 позначеної області інтересу. Дані були отримані у форматі Nifti1, щоб зберегти оригінальні величини одиниць Хаунсфілда. Розбиття вибірки даних на тренувальну, тестову та екзаменаційну вибірки представлені у таблиці 4.1.

Таблиця 3.1.

Розбиття вибірки

Клас	**Розмір тренувальної вибірки**	**Розмір тестової вибірки**	**Розмір екзаменаційної вибірки**	**Загальна кількість**
«Ground-glass opacity»	585	33	33	650
«Crazy-paving»	585	33	32	650
«Consolidation»	477	27	26	531

3.4 Підготовка даних та генерація ознак

Для класифікації типів ураження легень при COVID-19 будемо використовувати текстурні характеристики, які відображають характеристики залежності значень інтенсивності сусідніх пікселів зображення.

В цій роботі для кожної області інтересу були розраховані матриці GLCM з наступними просторовими співвідношеннями: (0, 1), (1, -1), (-1, 0). Окрім цього, були розраховані матриці GLRLM (Gray Level Run Length Matrix), GLSZM (Gray Level Size Zone Matrix), GLDM (Gray Level Dependence Matrix), NGTDM (Neighbouring Gray Tone Difference Matrix).

3.4.1 Структура нейронної мережі Feature-Constructor

Для генерації ознак був використаний конструктор ознак – сконфігурована згорткова нейронна мережа [10]. У роботі [3] схожу задачу вирішували за допомогою автокодера та аналізу головних компонент, проте для даної робота було застосовано саме перше рішення, оскільки основна перевага такого підходу полягає в тому, щоб отримати оптимальне формування ознак для найкращого вирішення розглянутої задачі класифікації.

Конструктор ознак (FC) має структуру, яка складається з кодувальника функцій текстури, агрегатора закодованих ознак і нейронного класифікатора. Кодер

текстурних особливостей являє собою набір із n нейронних мереж. Тут n – кількість обчислених матриць текстурних ознак. Кожному входу відповідає своя нейронна мережа. Згорткові нейронні мережі використовуються для значного зменшення кількості параметрів, які можна вивчати, порівняно з повністю зв’язаними мережами, зберігаючи при цьому високий рівень комбінування функцій. Запропоновано два паралельні типи мережевої структури для обробки характеристик текстури. Структура першого типу (рис. 3.2) була розроблена для обробки вхідних даних з матриць GLCM, GLRLM і GLSZM. Було запропоновано використовувати фільтри розміром 3x3 через відносно великі розміри цих матриць. В результаті значно меншого розміру матриць GLDM і NGTDM структура другого варіанту (рис. 3.3) розгортає матриці у вектор і обробляє їх за допомогою одновимірного фільтра 1x5.

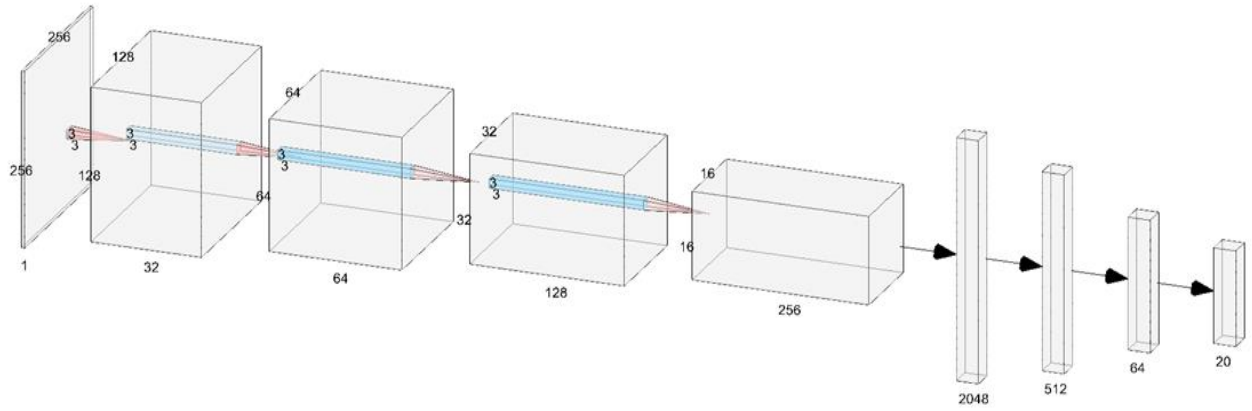

Рисунок 3.2 – Структура конструктору ознак для обробки GLCM, GLRLM і GLSZM.

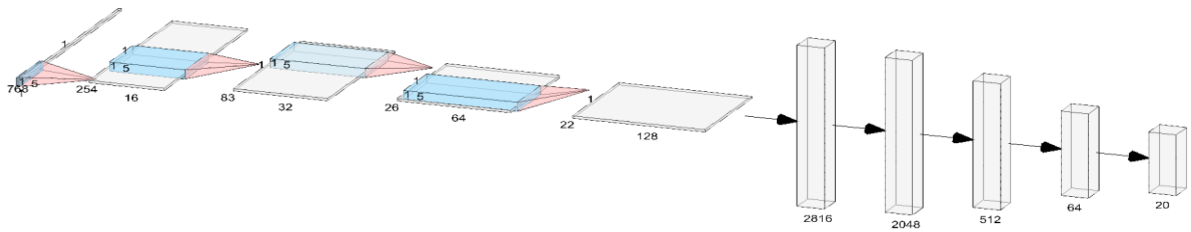

Рисунок 3.3 – Структура конструктору ознак для обробки GLDM і NGTDM.

Під час навчання всі частини Feature-Constructor були об’єднані в єдину нейронну мережу [10]. Основною метою навчання був пошук оптимальних параметрів, які мінімізують логарифмічні втрати або критерії крос-ентропії для функції класифікації. Серед цих параметрів є W_c та W_a, де W_c – ваги нейронних мереж кодувальників текстур, W_a – ваги нейронних мереж агрегаторів. Після навчання агрегатор формує орієнтовані на задані класи і оптимальні для цілей класифікації на ці класи, ознаки. Це дозволяє використати кодери та агрегатор без класифікатора та відкриває можливість створити гібридну структуру, замінивши частину класифікатора можливо більш ефективними алгоритмами.

3.5 Застосування самоорганізованого лісу.

Отримані в результаті конструювання ознаки, ставили вхідними даними для самоорганізованого логістичного лісу. Також була застосована техніка “skip connections”, за якою голоси дерев з моделі LSOF поєднуються з ознаками, отриманими від конструктору ознак з метою оптимізації функції логістичного голосування. Такий принцип поєднання ознак в структурах використовується в ResNet [15].

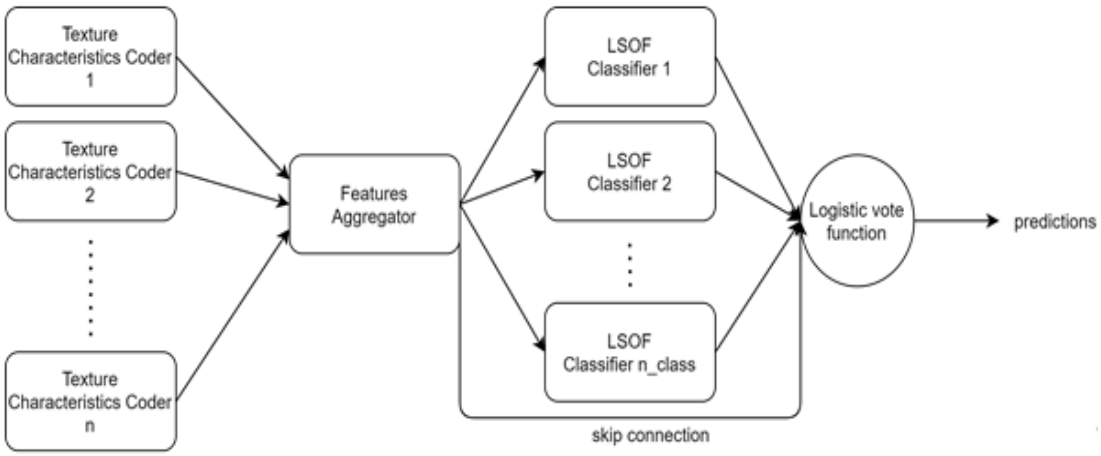

Рисунок 3.4 – структура розробленого гібридного класифікатора, де “Texture Characteristics Coder” – матриця GLCM, “Feature Aggregator” – конструктор ознак, “LSOF Classifier” – самоорганізоване дерево, “Logistic vote function” – логістична функція голосування

Поєднавши конструктор ознак та розроблений алгоритм, було отримано гібридний класифікатор, що здатний вирішувати задачу класифікації типів уражень легеней пацієнтів хворих на COVID-19 за областями інтересу. Отримана структура гібридного класифікатору зображена на рис. 3.4.

3.6 Приклади побудованих дерев

На рис. 3.5 наведено структуру одного із побудованих самоорганізованих дерев для q=1.

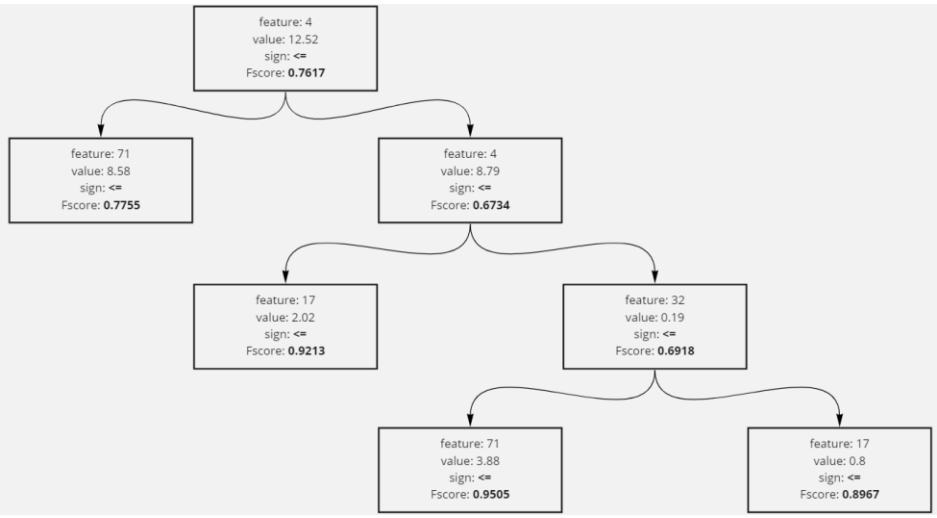

Рисунок 3.5 – структура побудованого самоорганізованого дерева для q=1

На рис. 3.6 наведено структуру одного із побудованих дерев самоорганізації для q=2.

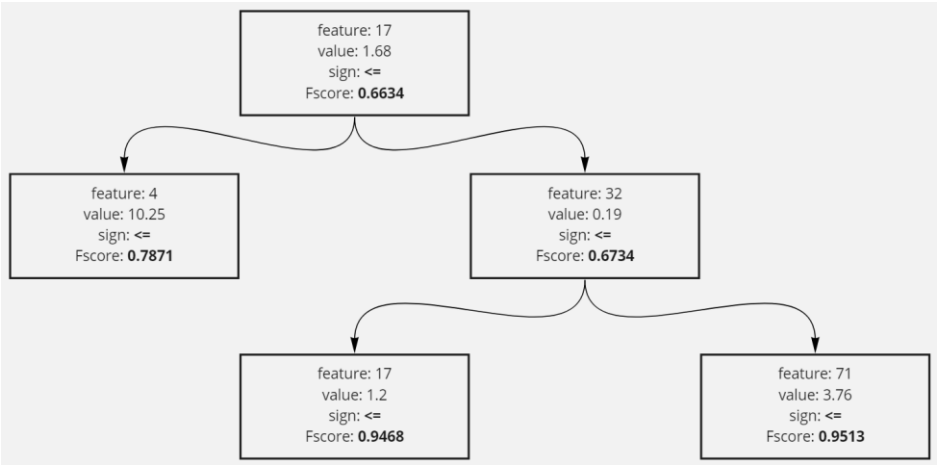

Рисунок 3.6 – структура побудованого самоорганізованого дерева для q=2

3.7 Порівняння результатів

Для оцінки роботи класифікатора, порівняємо наступні три результати прогнозування:

1. Класифікатор на основі нейронної мережі без гібридизації.
2. Гібридний класифікатор з конструктором ознак та базовою реалізацією випадкового лісу.
3. Гібридний класифікатор з конструктором ознак та алгоритмом самоорганізованізованого лісу.
4. Гібридний класифікатор з конструктором ознак та покращеним алгоритмом самоорганізованізованого лісу з глибини прогнозу q, де q=2.

Використання класифікаторів 1-4 дали результати, наведені в таблицях 3.2-3.5 відповідно.

Таблиця 3.2.

Результати першого класифікатора

Клас	**Точність**	**Повнота**	**F-метрика**
«ground-glass opacity»	0.92	0.92	0.92
«crazy-paving»	0.94	0.94	0.94
«consolidation»	0.91	0.91	0.91
Загальна точність	0.92		

Таблиця 3.3.

Результати другого класифікатора

Клас	**Точність**	**Повнота**	**F-метрика**
«ground-glass opacity»	0.92	0.92	0.92
«crazy-paving»	0.96	0.94	0.95
«consolidation»	0.91	0.91	0.91
Загальна точність	0.93		

Таблиця 3.4.

Результати третього класифікатора

Клас	**Точність**	**Повнота**	**F-метрика**
«ground-glass opacity»	1	1	1
«crazy-paving»	0.97	0.93	0.95
«consolidation»	0.91	0.96	0.93
Загальна точність	0.96		

Таблиця 3.5.

Результати четвертого класифікатора

Клас	**Точність**	**Повнота**	**F-метрика**
«ground-glass opacity»	1	1	1
«crazy-paving»	0.97	0.94	0.95
«consolidation»	0.92	0.96	0.94
Загальна точність	0.97		

Порівняння результатів загальної точності прогнозування наведені в таблиці 3.5.

Таблиця 3.5.

Порівняння моделей прогнозування

Модель	**Загальна точність**
Гібридний класифікатор із логістичним лісом самоорганізованих дерев із q=2	0.97
Гібридний класифікатор із логістичним лісом самоорганізованих дерев	0.96
Гібридний класифікатор із випадковим лісом	0.93
Класифікатор на основі нейронної мережі без гібридизації	0.92

Висновки до розділу 3

Розглянуто формування множини текстурних ознак для зображень комп'ютерної томографії: матриці GLCM GLRLM, GLSZM, GLDM, NGTDM. На одержаній множині ознак запропоновано формування гібридного класифікатору у складі згорткової нейронної мережі та удосконаленого самоорганізованого лісу для вирішення завдання класифікації типів ураження легень при захворюванні на Covid-19. Проведено порівняння точності запропонованих моделей класифікації.

Була вирішена задача класифікації типів уражень легеней з допомогою розробленого гібридного класифікатору. Проведено порівняння точностей моделей.

ЗАГАЛЬНІ ВИСНОВКИ

1. Обгрунтовано, розроблено та реалізовано алгоритм логістичного лісу самоорганізованих дерев за критерієм якості прогнозу класифікації. Алгоритм оптимізації структури дерев застосовує відбір ознак у вузли на поточному рівні дерева, що гарантуватимуть найкращий результат класифікації у вузлах на q рівнів глибше.

2. Побудовано гібридний класифікатор за участю моделі логістичного лісу самоорганізованих дерев для вирішення задачі класифікації типів уражень легень за областями інтересу на базі анонімних даних, що були надані ДУ «Національний інститут фтизіатрії та пульмонології імені Ф.Г. Яновського НАМН України». Порівняно результат точності прогнозування з іншими моделями, кращий результат одержано пропонованим алгоритмом

3. Розроблений гібридний класифікатор був впроваджений в практичну охорону здоров’я у ДУ «Національний інститут фтизіатрії та пульмонології імені Ф.Г. Яновського НАМН України» (Додаток Б).

4. Результати роботи відображено у публікації:

Анонімізовано / Hybrid Classifiers Based on CNN, LSOF, GMDH in COVID-19 Pneumonic Lesions Types Classification Task // Proceedings of the XVI IEEE International Conference CSIT-21& International Workshop on Inductive Modeling. Lviv, UKRAINE, 23-26 September, 2021 P. 380-384. ISSN Information:Electronic ISSN: 2766-3639 , Print on Demand(PoD) ISSN: 2766-3655, та запроваджено у медичну практику (наявний акт впровадження – Додаток А)

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. P. Silva, E. Luz, G. Silva, G. Moreira, R. Silva, D. Lucio, D. Menotti, “COVID-19 detection in CT images with deep learning: A voting-based scheme and cross-datasets analysis”, Informatics in Medicine Unlocked, Volume 20, 100427, ISSN 2352-9148 (2020).
2. V. Shah, R. Keniya, A. Shridharani, M. Punjabi, J. Shah, N. Mehendale, “Diagnosis of COVID-19 using CT scan images and deep learning techniques”, Emerg. Radiol. 28, 497–505 (2021).
3 Ş. Öztürk, U. Özkaya, M. Barstuğan, “Classification of Coronavirus (COVID-19) from X-ray and CT images using shrunken features”, International journal of imaging systems and technology, 10.1002/ima.22469. Advance online publication (2020).
4 E. D. Carvalho, E. D. Carvalho, A. O. de Carvalho Filho, F. H. D. de Araújo and R. d. Andrade Lira Rabêlo, "Diagnosis of COVID-19 in CT image using CNN and XGBoost," 2020 IEEE Symposium on Computers and Communications (ISCC), 2020, pp. 1-6, doi: 10.1109/ISCC50000.2020.9219726.
5 I. Nastenko, V. Maksymenko, A. Galkin, V. Pavlov, O. Nosovets, I. Dykan, B. Tarasiuk, V.Babenko, V. Umanets, O. Petrunina, D. Klymenko, “Liver Pathological States Identification with Self-organization Models Based on Ultrasound Images Texture Features,” In: Shakhovska N., Medykovskyy M.O. (eds) Advances in Intelligent Systems and Computing V. CSIT 2020. Advances in Intelligent Systems and Computing, vol 1293. Springer, Cham (2021).
6 P. Varalakshmi, V. Narayanan, Sakthi Jaya Sundar Rajasekar, “Detection of COVID-19 using CXR and CT images using Transfer Learning and Haralick features,” Applied Intelligence. 51. 10.1007/s10489-020-01831-z (2021).
7 Tin Kam Ho Random decision forests. IEEE Comput. Soc. Press, 1995.

8 Prediction of Lung Cancer Risk using Random Forest Algorithm Based on Kaggle Data Set. International Journal of Recent Technology and Engineering. 2020. Vol. 8, No. 6. pp. 1623–1630.

9 Anastasakis, L.; Mort, N. The development of self-organization techniques in modelling: a review of the group method of data handling (GMDH). RESEARCH REPORT-UNIVERSITY OF SHEFFIELD DEPARTMENT OF AUTOMATIC CONTROL AND SYSTEMS ENGINEERING, 2001.

10 Davydko, O., Hladkyi, Y., Linnik, M., Nosovets, O., Pavlov, V., & Nastenko, I. (2021, September). Hybrid Classifiers Based on CNN, LSOF, GMDH in COVID-19 Pneumonic Lesions Types Classification Task. In 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 380-384). IEEE.

11 Ivakhnenko, A. G. "Heuristic self-organization in problems of engineering cybernetics". Automatica, 1970, 6.2: 207-219.

12 Ivakhnenko A.G. "The group method of data handling – a rival of the method of stochastic approximation", Soviet Automatic Control c/c of Avtomatika, vol.1, no.3, pp.43-55, 1968.

13 Lavalley, Michael P. Logistic regression. Circulation, 2008, 117.18: 2395-2399.

14 Hall-Beyer, Mryka. GLCM texture: a tutorial. National Council on Geographic Information and Analysis Remote Sensing Core Curriculum, 2000, 3.1: 75.

15 K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition," In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778) (2016).

Додаток А.

«ЗАТВЕРДЖУЮ»
Головний лікар ДУ «Національний інститут фтизіатрії і пульмонології ім. Ф.Г. Яновського НАМН України»
проф., д.мед.н. Микола Опанасенко
«15» листопада 2022 р.

Акт

впровадження в практичну охорону здоров'я матеріалів наукових досліджень

Найменування пропозиції для впровадження:

1. Система класифікації для визначення типів уражень (матове скло, бруківка, консолідація) легень по КТ-зображенням при захворюванні на Covid-19
2. Система класифікації для диференціювання по КТ-зображенням легень хіміорезистентної та хіміочутливої форми туберкульозу.

Ким запропоновано: НТУУ «Київський політехнічний інститут імені Ігоря Сікорського»,

Джерело інформації:

1. Матвійчук О.В. Сегментація туберкульозних уражень легень на зображеннях комп'ютерної томографії / О.В. Матвійчук, С.Н. Ворончук. К.С.Бовсуновська, О.Б/ Давидько, М.І. Линник, А.В. Павлов, С.А. Настенко // Innovative Biosystems and Bioengineering, (2021), 5(2) https://doi.org/10.20535/ibb.2021.5.2.233051
2. Давидько О.Б., , Класифікація уражень легень при covid-19 на основі текстурних ознак та з горткової нейронної мережі. / О.Б. Давидько, А.О. Ладік, В.Б. Максименко, М. І. Линник, О.В. Павлов, Є.А. Настенко // Біомедична інженерія і технологія (2021), №6 https://doi.org/ 10.20535/2617-8974.2021.6.231887
3. Davydko O., Hladkyi Y., Linnik M., Nosovets O., Pavlov V., Nastenko Ie. / Hybrid Classifiers B ased on CNN, LSOF, GMDH in COVID-19 Pneumonic Lesions Types Classification Task // Pro ceedings of the XVI IEEE International Conference CSIT-21& International Workshop on Induct ive Modeling. Lviv, UKRAINE, 23-26 September, 2021 P. 380-384. DOI: 10.1109/CSIT52700.2 021.9648752
4. Matviichuk O., Nosovets O., Linnik M., Davydko O., Pavlov V., Nastenko Ie. / Class-Oriented F eatures Selection Technology in Medical Images Classification Problem on the Example of Distin guishing Between Tuberculosis Sensitive and Resistant Forms// Proceedings of the XVI IEEE Int ernational Conference CSIT-21& International Workshop on Inductive Modeling. Lviv, UKRAI NE, 23-26 September, 2021 P. 385-389. DOI: 10.1109/CSIT52700.2021.9648747

Впроваджено

1. Система класифікації для визначення типів уражень (матове скло, бруківка, консолідація) легень по КТ-зображенням при захворюванні на Covid-19
2. Система класифікації для диференціювання по КТ-зображенням легень хіміорезистентної та хіміочутливої форми туберкульозу

Строки впровадження: листопад 2022р.

Завідуючий відділом
епідеміологічних та організаційних проблем
фтизіопульмонології ДУ «Національний інститут
фтизіатрії і пульмонології ім. Ф.Г. Яновського
НАМН України», д.мед.н Микола Линник

Додаток В.

```
from itertools import combinations
from typing import List, Tuple

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split as sk_split
from sklearn.metrics import confusion_matrix, f1_score
from sklearn.linear_model import LogisticRegression

from core.features.generation import feature_generate

from loguru import logger
from tqdm import tqdm


def split_data(X, y, **kwargs):
    X_train, X_test, y_train, y_test = sk_split(X, y, stratify=y, **kwargs)
    return X_train, X_test, y_train, y_test

class Dataset():
    def __init__(self, data_train, data_test, y_train, y_test):
        self.CD_X = data_test
        self.CD_y = y_test
        self.AB_X, self.C_X, self.D_X, self.AB_y, self.C_y, self.D_y =
self._load_data(data_train, data_test, y_train, y_test)

    def _load_data(self, data_train, data_test, y_train, y_test):
        # data = pd.read_csv(data_file)
        data_train = data_train.apply(pd.to_numeric, downcast='float')
        # split CD into C and D
        C_X, D_X, C_y, D_y = split_data(data_test, y_test, test_size=0.5, random_state=42)
        return data_train, C_X, D_X, y_train, C_y, D_y

    def get_CD(self):
        return np.concatenate((self.C_X, self.D_X)), np.concatenate((self.C_y, self.D_y))

    def split_AB(self, is_random=True):
        '''
        Random A and B sets for a tree
        '''

        if is_random:
            test_size=0.3 + (np.random.random() - 0.5) / 10
            A_X, B_X, A_y, B_y = split_data(self.AB_X, self.AB_y, test_size=test_size)
        else:
            test_size=0.3
            A_X, B_X, A_y, B_y = split_data(self.AB_X, self.AB_y, test_size=test_size,
random_state=42)
```

```
        return A_X, B_X, A_y, B_y

class FScore():
    @staticmethod
    def get(real, predictions, criteria='f1'):
        '''
        Score to compare features
        '''
        try:
            np_pred = predictions if isinstance(predictions, np.ndarray) else
predictions.to_numpy()
            np_real = real.to_numpy()
            where_equal = np_pred == np_real
            where_not_equal = np_pred != np_real
            differentiator = np_pred[where_not_equal] - np_real[where_not_equal]
            modified_real = np_real[where_equal].astype(np.int)
            modified_real[modified_real == 0] = 2
            t_differentiator = np_pred[where_equal] - modified_real
            tp = np.count_nonzero(t_differentiator == 0)
            tn = np.count_nonzero(t_differentiator == -2)
            fn = np.count_nonzero(differentiator == -1)
            fp = np.count_nonzero(differentiator == 1)
            # if no errors
            if fp == fn == 0:
                raise ValueError
        # if all values matched
        except ValueError:
            return 1

        if criteria == 'f1':
            f1_1 = tp / (tp + (fp + fn) / 2)
            f1_0 = tn / (tn + (fn + fp) / 2)
            return np.mean((f1_1, f1_0))
        elif criteria == 'sensitivity':
            recall_1 = tp / tp + fn
            recall_0 = tn / tn + fp
            return np.mean((recall_1, recall_0))
        else:
            raise KeyError(f'Undefined criteria: {criteria}')


class Node:
    def __init__(self, best_n_features, decision=None):
        self.best_n_features = best_n_features
        self.feature, self.threshold, self.less_or_equal_sign, self.fscore_AB = max(
            best_n_features, key=lambda best_feature: best_feature[-1]
        )
        self.true_node = self.false_node = None

    def get_props(self):
        return (self.feature, self.threshold, self.less_or_equal_sign, self.fscore_AB)

    def set_props(self, props):
```

```
        print(props)
        feature, threshold, less_or_equal_sign, fscore_AB = props
        self.feature, self.threshold, self.less_or_equal_sign, self.fscore_AB = props



def visualize(tree):
    def print_node(node, level):
        if node == None:
            return

        print_node(node.true_node, level + 1)
        print('\n' + '\t' * level + 'node:', node.feature + '(', node.threshold, ',',
              '<=' if node.less_or_equal_sign else '>', ')', 'fscore =,', node.fscore_AB)
        print_node(node.false_node, level + 1)

    print_node(tree.root, 0)



class Tree:
    def __init__(self, A_X, B_X, A_y, B_y):
        self.A_X, self.B_X, self.A_y, self.B_y = A_X, B_X, A_y, B_y
        self.AB_X = pd.concat((self.A_X, self.B_X))
        self.AB_y = pd.concat((self.A_y, self.B_y))
        self.features = A_X.columns
        self.root = None

    def __get_data_by_index(self, data, idxs):
        try:
            return data.loc[idxs]
        except KeyError:
            return data.loc[idxs.intersection(data.index)]

    def _find_best_feature_threshold(self, node_idxs: pd.Index, feature_name: str) ->
Tuple[float, bool]:
        '''
        Find best threshold for given feature CONSIDERING sign.
        Selecting best threshold for feature on A(!) set.
        '''
        # get data that go to this node
        node_X = self.__get_data_by_index(self.A_X, node_idxs)
        node_y = self.__get_data_by_index(self.A_y, node_idxs)

        feature_vector = node_X[feature_name]

        center_1 = np.mean(feature_vector[node_y == 1])
        center_0 = np.mean(feature_vector[node_y == 0])



        # get set of all candidates
        if center_1 > center_0:
```

```
            threshold_candidates = np.unique(feature_vector[np.logical_and(feature_vector >=
center_0, feature_vector <= center_1)])
        else:
            threshold_candidates = np.unique(feature_vector[np.logical_and(feature_vector >=
center_1, feature_vector <= center_0)])

        if threshold_candidates.size == 0:
            threshold_candidates = np.unique(feature_vector)

        # if only one value exists then return it
        if threshold_candidates.size == 1:
            return (threshold_candidates[0], True)

        # get centers of thresholds
        threshold_candidates = threshold_candidates[:-1] + (threshold_candidates[1:] -
threshold_candidates[:-1]) / 2

        # best parameters
        best_f_score = -np.inf
        best_threshold_with_sign = (None, None)

        # compare all thresholds
        for threshold in threshold_candidates:
            # try with different signs
            for less_or_equal_sign in True, False:
                # node.less_or_equal_sign = less_or_equal_sign

                candidate_f_score = FScore.get(
                    node_y,
                    feature_vector <= threshold if less_or_equal_sign else feature_vector >
threshold
                )

                # current parameters beat the record
                if candidate_f_score > best_f_score:
                    best_f_score = candidate_f_score
                    best_threshold_with_sign = (threshold, less_or_equal_sign)

        return best_threshold_with_sign

    def _find_best_features(self, node_idxs: pd.Index, n: int = -1) -> List[Tuple[str,
float, bool, float]]:
        '''
        Calculate best n features for given dataset.
        Selecting best threshold for feature on AB(!) set.
        '''
        A_X = self.__get_data_by_index(self.A_X, node_idxs)
        AB_X = self.__get_data_by_index(self.AB_X, node_idxs)
        AB_y = self.__get_data_by_index(self.AB_y, node_idxs)

        # get all candidates from those that saved in node
        # features_candidates = node.best_n_features
        # if this list is empty or None, then use all features in the dataset
```

```
        # if not features_candidates:
        features_candidates = self.features

        # get array of pairs: (feature threshold, less or equal/more sign indicator)
        thresholds_with_sign = [
            self._find_best_feature_threshold(node_idxs, feature_name) for feature_name in
features_candidates
        ]

        fscores_AB = [
            FScore.get(
                AB_y,
                AB_X[feature] <= threshold if less_or_equal_sign else AB_X[feature] >
threshold
            )
            if threshold != None else 0
            for feature, (threshold, less_or_equal_sign) in zip(features_candidates,
thresholds_with_sign)
        ]

        if n == -1:
            n = self.features.size

        # get indicies of top n values
        best_n_features_idxs = np.argpartition(fscores_AB, -n)[-n:]

        # return list of tuples:
        # (feature, threshold for given feature, sign for given threshold, F score on AB
set)
        return [
            (
                features_candidates[feature_idx],
                *thresholds_with_sign[feature_idx],
                fscores_AB[feature_idx]
            ) for feature_idx in best_n_features_idxs
        ]

    def get_tree_f_score(self, on_A_set=False):
        predictions = self.classify_data(
            self.A_X if on_A_set else self.AB_X
        )
        return FScore.get(self.AB_y, predictions)

    def build_node(self, node, node_idxs, best_n_child_features=10, depth=1):
        best_sum_children_f_scores = -np.inf
        true_node_data_idxs = None
        false_node_data_idxs = None
        node_less_or_equal_sign = None
        node_true_best_n_features = None
        node_false_best_n_features = None

        true_node_to_configure = True
        false_node_to_configure = True
```

```
        AB_X = self.__get_data_by_index(self.AB_X, node_idxs)

        print(node.best_n_features)

        for (feature_candidate, feature_candidate_threshold, threshold_sign,
feature_f_score) in node.best_n_features:
            #             if feature_candidate_threshold == None:

            # AB_X is a pandas Series, so its index can be used for both A and AB sets
            true_idxs = (AB_X[
                AB_X[feature_candidate] <= feature_candidate_threshold
                if threshold_sign
                else AB_X[feature_candidate] > feature_candidate_threshold
            ]).index
            false_idxs = AB_X.index.difference(true_idxs)

            true_best_n_features = self._find_best_features(true_idxs,
best_n_child_features)
            false_best_n_features = self._find_best_features(false_idxs,
best_n_child_features)

            true_node_best_f_score = np.max([f_score for _, _, _, f_score in
true_best_n_features])
            false_node_best_f_score = np.max([f_score for _, _, _, f_score in
false_best_n_features])

            # check if ready node decision will be better
            true_node_total_ones_f_score = FScore.get(
                self.__get_data_by_index(self.AB_y, true_idxs),
                np.ones(true_idxs.shape)
            )
            false_node_total_zeros_f_score = FScore.get(
                self.__get_data_by_index(self.AB_y, false_idxs),
                np.zeros(false_idxs.shape)
            )

            if true_node_best_f_score < true_node_total_ones_f_score:
                true_node_best_f_score = true_node_total_ones_f_score

            if false_node_best_f_score < false_node_total_zeros_f_score:
                false_node_best_f_score = false_node_total_zeros_f_score

            #             logger.debug(f'{feature_candidate=}, {feature_f_score=},
{threshold_sign=}')

            if true_node_best_f_score + false_node_best_f_score >
best_sum_children_f_scores:
                best_sum_children_f_scores = true_node_best_f_score +
false_node_best_f_score
                logger.debug(f'best children: {feature_candidate}: {true_node_best_f_score},
{false_node_best_f_score}')
                logger.debug(f'feature candidate: ')
```

```
                node.feature = feature_candidate
                node.threshold = feature_candidate_threshold
                node.fscore_AB = feature_f_score
                node.less_or_equal_sign = threshold_sign
                true_node_data_idxs = true_idxs
                false_node_data_idxs = false_idxs
                node_true_best_n_features = true_best_n_features
                node_false_best_n_features = false_best_n_features

        next_nodes_to_configure = []

        if true_node_best_f_score != 1:
            node.true_node = Node(node_true_best_n_features)
            next_nodes_to_configure.append((node.true_node, true_node_data_idxs, depth + 1))

        if false_node_best_f_score != 1:
            node.false_node = Node(node_false_best_n_features)
            next_nodes_to_configure.append((node.false_node, false_node_data_idxs, depth +
1))

        return next_nodes_to_configure

    #         print(node_true_best_n_features, node_false_best_n_features)

    #         node.true_node = Node(node_true_best_n_features)
    #         node.false_node = Node(node_false_best_n_features)
    #         return  [
    #             (node.true_node, true_node_data_idxs), (node.false_node,
false_node_data_idxs)
    #         ]

    def classify(self, node, X, classified):
        if node is None:
            return

        true_subX = X[X[node.feature] <= node.threshold]
        false_subX = X[X[node.feature] > node.threshold]

        if not node.less_or_equal_sign:
            true_subX, false_subX = false_subX, true_subX

        classified.loc[true_subX.index] = 1
        classified.loc[false_subX.index] = 0

        self.classify(node.true_node, true_subX, classified)
        self.classify(node.false_node, false_subX, classified)

    def classify_data(self, X):
        if X.ndim == 2:
            classified = pd.Series(np.zeros(X.shape[0]), index=X.index)
            self.classify(self.root, X, classified)
            return classified
```

```
        else:
            raise RuntimeError("Data to classify must be 2-dimensional")

    def train(self, tolerance=0, max_depth=9):
        # create root with all features
        self.root = Node(self._find_best_features(self.AB_X.index))
        configuration_queue = self.build_node(self.root, self.AB_X.index)
        tree_f_score = self.get_tree_f_score()
        logger.info(f'Tree F-score: {tree_f_score}')

        while configuration_queue:
            current_node, current_node_idxs, depth = configuration_queue.pop(0)

            if depth > max_depth:
                continue

            # save current state of the node
            saved_props = current_node.get_props()

            # get new children after building node
            new_nodes = self.build_node(current_node, current_node_idxs, depth=depth)
            new_tree_f_score = self.get_tree_f_score()

            visualize(self)

            # logger.debug(current_node_idxs)
            # logger.debug(f'configureation node: {current_node.feature}')
            logger.info(f'old tree FScore: {tree_f_score}')
            logger.info(f'new tree FScore: {new_tree_f_score}')

            # if the tree did not become better
            if new_tree_f_score - tree_f_score <= tolerance:
                # return node its state
                current_node.set_props(saved_props)
                # and remove its children
                current_node.true_node = None
                current_node.false_node = None
            else:
                tree_f_score = new_tree_f_score
                configuration_queue.extend(new_nodes)


class LogisticSelfOrganizedForest():
    def __init__(self, dataset):
        self.trees = []
        self.dataset = dataset
        self.lr_model = LogisticRegression()

    def train(self, multithreading=True, tolerance=2):
        tolerance_counter = 0.0
        best_metric = 0.0
        while True:
            x_a, x_b, y_a, y_b = self.dataset.split_AB()
```

```
            new_tree = Tree(x_a, x_b, y_a, y_b)
            self.trees.append(new_tree)
            new_tree.train()
            trees_votes_AB = np.array([tree.classify_data(self.dataset.AB_X) for tree in
self.trees]).T
            trees_votes_CD = np.array([tree.classify_data(pd.DataFrame(self.dataset.CD_X))
for tree in self.trees]).T
            self.lr_model = LogisticRegression(max_iter=1000000)
            self.lr_model.fit(trees_votes_AB, self.dataset.AB_y)
            current_metric = self.lr_model.score(trees_votes_CD, self.dataset.CD_y)
            logger.info(f'current overall metric: {current_metric}')
            if current_metric > best_metric:
                logger.info(f'new best metric!')
                best_metric = current_metric
                tolerance_counter = 0
            elif (current_metric <= best_metric):
                tolerance_counter += 1
            if tolerance_counter == tolerance:
                break

class LogisticSelfOrganizedForestClassifier():
    def __init__(self, n_best_features=10):
        self.trees = []
        self.n_best_features = n_best_features
        self.best_features_indices = {}
        self.lr_model = LogisticRegression()


    def _get_trees_votes(self, X):
        return np.array([tree.classify_data(X) for tree in self.trees]).T

    def fit(self, X_train, y_train, X_dev, y_dev):
        # Inductively augment features
        forests = []
        train = X_train
        dev = X_dev
        #train = feature_generate(X_train, X_train.columns, division=None)
        #train.replace([np.inf, -np.inf], np.nan, inplace=True)
        #train = train.fillna(0)

        #dev = feature_generate(X_dev, X_dev.columns, division=None)
        #dev.replace([np.inf, -np.inf], np.nan, inplace=True)
        #dev.fillna(0)
        unique_labels = np.unique(y_train)
        if unique_labels.shape[0] > 2:
            # Handle multiclass task
                for i in unique_labels:
                    print(f'Class: {i}')
                    labels_train = np.zeros_like(y_train)
                    labels_train[y_train == i] = 1

                    best_features_indices = np.argsort(train.corrwith(y_train,
method='kendall').to_numpy())[::-1][:self.n_best_features]
```

```
                    self.best_features_indices[i] = best_features_indices

                    labels_dev = np.zeros_like(y_dev)
                    labels_dev[y_dev == i] = 1
                    dataset = Dataset(train.iloc[:, best_features_indices], dev.iloc[:,
best_features_indices], pd.Series(labels_train), pd.Series(labels_dev))
                    forest = LogisticSelfOrganizedForest(dataset)
                    forest.train()
                    forests.append(forest)
        self.trees = []
        for f in forests:
            self.trees += f.trees
        trees_votes = self._get_trees_votes(train)
        lr_train_data = np.concatenate([train, trees_votes], axis=1)
        self.lr_model = LogisticRegression(max_iter=1000000)
        self.lr_model.fit(lr_train_data, y_train)
        return self

    def predict(self, X):
        augmented = feature_generate(X, X.columns, division=None)
        trees_votes = self._get_trees_votes(X)
        lr_predict_data = np.concatenate([X, trees_votes], axis=1)
        return self.lr_model.predict(lr_predict_data)




class ForestClasifier():
    def __init__(self, forests):
        self.forests = forests
        trees = []
        for f in self.forests:
            trees += f.trees
        self.trees = trees

    def classify(self, data):
        return np.array([
            tree.classify_data(data)
            for tree in self.trees
        ]).mean(axis = 0)

    def get_logits(self, data):
        logits = []
        for f in self.forests:
            logits += f.rl_model.predict_proba(data)
        return np.concat(logits, axis=1)
```

Анотація

Робота складається зі: вступу, 3 розділів, висновків до кожного з цих розділів, загальних висновків, списку літературних джерел, додатку. Загальний обсяг роботи – 29 сторінок (без врахування додатку та списку використаних джерел).

Актуальність: Пандемія COVID-19 стала однією з найяскравіших глобальних реакцій на безперервне зростання людської популяції. Сьогодні потрібні ефективні діагностичні та аналітичні інструменти для боротьби з наслідками інфекції. При цьому оцінка стану хворого в значній мірі визначається кількісними показниками різного типу ураження легень (матове скло, бруківка, консолідація). В роботі розглядається застосування гібридної системи класифікації на основі згорткової нейронної мережі та удосконаленого алгоритму самоорганізованого лісу для вирішення задачі класифікації типів уражень легень при Covid-19 на знімках комп'ютерної томографії.

Мета роботи: Підвищення точності систем класифікації типів уражень легень при Covid-19 на знімках комп'ютерної томографії.

Задачі роботи:

• Розробка алгоритму та програмної реалізації для формування класифікатору на основі удосконаленого алгоритму самоорганізованого лісу.

• Розробка гібридної системи класифікації типів уражень легень при Covid-19 на основі згорткової нейронної мережі та самоорганізованого лісу

Публікації:

Результати роботи доповідались та були опубліковані у "Proceedings of the XVI IEEE International Conference CSIT-21& International Workshop on Inductive Modeling" (Scopus).

Використана методика:

Запропоновано формувати дерева моделі класифікації «Ліс» за принципами самоорганізації. Структура дерев на поточному m-тому рівні формується за зовнішнім критерієм, що досягає найкращого значення на (m+k)-тому рівні дерева,

де k - глибина упередження. Оптимізація структури функції голосування дерев лісу здійснюється за рахунок врахування первинних ознак, застосування логістичного перетворення та розрахунку оптимальних значень параметрів функції голосування. Апробацію запропонованого алгоритму здійснено при формуванні гібридного класифікатору на основі згорткової нейронної мережі та самоорганізованого лісу. Класифікатор застосовано для вирішення задачі класифікації на знімках комп'ютерної томографії областей типів уражень легень при захворюванні на Covid-19. Результати роботи запроваджено у медичну практику. Наявний акт впровадження.

Ключові слова: Ліс, гібридний класифікатор, нейронна мережа, текстурні ознаки, самоорганізація, COVID-19, комп'ютерна томографія, легені.