Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка
до дипломного проекту (роботи)
магістра
(освітньо-кваліфікаційний рівень)

на тему Розроблення програмного забезпечення для контролю та підвищення ефективності нейромережевих експертних систем

Виконав: студент 6 курсу, групи 602-ТН
спеціальності
122 Комп'ютерні науки та інформаційні технології
(шифр і назва напряму)
Білокінь О.В.
(прізвище та ініціали)
Керівник Альошин С.П.
(прізвище та ініціали)
Рецензент Горішній Д.О.
(прізвище та ініціали)

Полтава – 2024 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

спеціальність 122 «Комп'ютерні науки та інформаційні технології»

на тему

«Розроблення програмного забезпечення для контролю та підвищення ефективності нейромережевих експертних систем»

Студента групи 602-ТН Білоконя Олега Владиславовича

Керівник роботи
кандидат технічних наук,
доцент Альошин С.П.

Завідувач кафедри
кандидат фізично-математичних наук,
доцент Двірна О. А.

Полтава – 2024

РЕФЕРАТ

Склад. Пояснювальна записка до проекту налічує 69 с., 3 табл., 30 рисунків, 30 джерел інформації.

Анотація. Експертні системи є важливою складовою сучасних аналітичних процесів. Створення програмного рішення для контролю та оптимізації налаштування експертних систем може значно покращити продуктивність роботи з ними. Scikit-Learn – це бібліотека функцій для навчання нейромережевих моделей. Створене програмне забезпечення забезпечує редагування тренувальних наборів, дозволяє обрати оптимальні параметри тренування, візуалізувати статистику.

Мета. Створити систему контролю та підвищення ефективності нейромережевих експертних систем.

Ключові слова: експертна система, нейромережеві моделі, глибоке навчання, класифікатори, моделі регресії, Python.

REFERENCE

Summary. The work consists of 69 pages, 3 tables, 30 figures, 30 sources of information.

Abstract. Expert systems are an important component of modern analytical processes. Creating a software solution to control and optimize the configuration of expert systems can significantly improve their performance. Scikit-Learn is a library of functions for training neural network models. The software allows editing training sets, selecting optimal training parameters, and visualizing statistics.

Objective. Create a system for monitoring and improving the efficiency of neural network expert systems.

Keywords: expert system, neural network models, deep learning, classifiers, regression models, Python..

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект.

ПП – програмний продукт

IDE (англ. Integrated development environment) – комплексне програмне рішення для розробки програмного забезпечення.

ОС – операційна система.

CUDA (англ. Compute Unified Device Architecture) — програмно-апаратна архітектура паралельних обчислень, яка дозволяє істотно збільшити обчислювальну продуктивність завдяки використанню графічних процесорів фірми.

API (англ. Application Programming Interface) — набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. API надає розробнику засоби для швидкої розробки програмного забезпечення.

SDK – (від англ. software Development Kit) — набір із засобів розробки, утиліт і документації, який дозволяє програмістам створювати прикладні програми за визначеною технологією або для певної платформи (програмної чи програмно-апаратної).

Python – інтерпретована, об’єктно-орієнтована мова програмування високого рівня з динамічною семантикою.

Нейронна мережа – є підмножиною машинного навчання і лежать в основі алгоритмів глибокого навчання. Нейронні мережі є серіями алгоритмів, які намагаються розпізнати основні взаємозв’язки в наборі даних за допомогою процесу, що імітує роботу людського мозку.

CNN (англ. Convolutional Neural Network, укр. Згорткова Нейронна Мережа)– це клас штучних нейронних мереж, які найчастіше застосовуються для аналізу візуальних зображень. ЗНМ використовують математичну операцію під назвою згортка замість звичайного множення матриць принаймні в одному зі

своїх шарів. Вони спеціально розроблені для обробки піксельних даних і використовуються в розпізнаванні та обробці зображень.

Нейрон – в моделях глибокого навчання, вузол, через який проходять дані та обчислення.

Шар – це компонент моделі найвищого рівня в машинному навчанні. Шар – це контейнер, який зазвичай отримує зважені вхідні дані, перетворює їх за допомогою набору переважно нелінійних функцій, а потім передає ці значення на вихід наступному шару.

Функція активації – це функція, яка додається до шару штучної нейронної мережі, щоб допомогти мережі вивчати складні закономірності в даних.

Softmax (нормована експоненційна функція)– це функція, яка перетворює вектор з K дійсних значень у вектор з K дійсних значень, сума яких дорівнює 1.

ВСТУП

Продуктивність є найважливішою якістю сучасних IT-систем і може розглядатися з різних точок зору, таких як пропускна здатність, час відгуку, використання ресурсів тощо. Проблеми з продуктивністю, починаючи від погіршення показників продуктивності і закінчуючи системними збоями, можуть призвести до незадоволення клієнтів та кінцевих користувачів і, як наслідок, до фінансових втрат для зацікавлених сторін. Згідно зі статистикою Dun&Bradstree, 59% компаній зі списку Fortune 500 зазнають простоїв тривалістю 1,7 години на тиждень, що, за прогнозами, становить 46 мільйонів доларів США щорічно для компаній з кількістю працівників понад 10 000 осіб [18]. Насправді, згідно з різними дослідженнями [20], існує більша ймовірність того, що система вийде з ладу через проблеми з продуктивністю, ніж через функціональні.

Тестування продуктивності - це процес визначення того, як система працює з точки зору стабільності та швидкості реагування в умовах підвищеного навантаження. Кінцевою метою тестування продуктивності є пошук та усунення дефектів продуктивності [19], [20] шляхом створення та виконання тестів проти системи та спостереження за продуктивністю системи за допомогою ключових показників ефективності (KPI). На відміну від функціонального тестування, де тести призначені для виявлення помилок, які проявляються в неправильному виведенні, при тестуванні продуктивності ми зацікавлені в тому, щоб знайти тести, які змусять KPI перевищити цільові значення.

Проведення тестування продуктивності на практиці є складним завданням не тільки тому, що воно зазвичай виконується на останніх етапах процесу розробки, коли система зазвичай розглядається як чорний ящик, але й тому, що в складних системах кількість можливих тестових вхідних даних може бути дуже великою, що зробить тестування системи вичерпним непрактичним. Останнє особливо спостерігається в системах, які мають багато конфігураційних параметрів, що призводить до проблеми комбінаторного вибуху.

Класифікація – одна з найпоширеніших задач прийняття рішень у людській діяльності. Проблема класифікації виникає тоді, коли об’єкт потрібно віднести до заздалегідь визначеної групи або класу на основі ряду спостережуваних атрибутів, пов’язаних з цим об’єктом. Багато проблем у бізнесі, науці, промисловості та медицині можна розглядати як проблеми класифікації. Приклади включають прогнозування банкрутства, обрахування кредитного рахунку, медичну діагностику, контроль якості, розпізнавання рукописних символів і розпізнавання мови.

Традиційні процедури статистичної класифікації, такі як дискримінантний аналіз, побудовані на байєсівській теорії прийняття рішень. [1] У цих процедурах для обчислення апостеріорної ймовірності, на основі якої приймається рішення про класифікацію, необхідно припустити базову ймовірнісну модель. Одним з основних обмежень статистичних моделей є те, що вони добре працюють лише тоді, коли задовольняються основні припущення. Ефективність цих методів значною мірою залежить від різних припущень або умов, за яких розробляються моделі. Користувачі таких моделей повинні добре знати як властивості даних, так і можливості моделей, перш ніж їх можна буде успішно застосовувати.

Нейронні мережі стали важливим інструментом для задач прогнозування, аналізу, генерування медіа [2] (додаток Б) та класифікації. Нещодавні масштабні дослідження в галузі нейронної класифікації показали, що нейронні мережі є багатообіцяючою альтернативою різним традиційним методам класифікації. Перевага нейронних мереж полягає в наступних теоретичних аспектах.

По-перше, нейронні мережі є самоадаптивними методами, керованими даними, оскільки вони можуть підлаштовуватися під дані без будь-якої явної специфікації функціональної або дистрибутивної форми для базової моделі.

По-друге, вони є універсальними функціональними апроксиматорами, оскільки нейронні мережі можуть апроксимувати будь-яку функцію з довільною точністю. Оскільки будь-яка процедура класифікації шукає функціональну залежність між приналежністю до групи та атрибутами об’єкта, то точна

ідентифікація цієї базової функції, безсумнівно, є важливою.

По-третє, нейронні мережі є нелінійними моделями, що робить їх гнучкими у моделюванні складних взаємозв'язків реального світу. Нарешті, нейронні мережі здатні оцінювати апостеріорні ймовірності, що є основою для встановлення правил класифікації та проведення статистичного аналізу. [1]

Актуальність теми. В останні часи, медичні заклади зіштовхнулись з понаднормовими об'ємами завдань через неочікуване збільшення кількості нових пацієнтів та нестабільність робочого середовища. Комп'ютеризація робочого процесу може значно знизити навантаження на працівників, що дозволить покращити ефективність роботи та знизити ризики пов'язані з перенавантаженням. [3]

Метою даної роботи є побудова нейромережевого класифікатора медичних знімків для підтримки ефективної роботи закладів охорони здоров'я в сучасних умовах.

Виходячи з мети, у роботі постають наступні завдання:

- окреслити проблематику створення та оптимізації моделей регресії та класифікації;
- оглянути існуючі рішення, визначити переваги та недоліки для цих рішень;
- сформулювати вимоги до програмного забезпечення;
- розробити модель нейромережевого класифікатора, дослідити метрику роботи моделі;
- розробити програмне забезпечення для зниження складності процесу налаштування та тренування моделі;
- провести випробування розробленого ПЗ.

Предметом дослідження є системи контролю та підвищення ефективності експертних систем.

Об'єктом дослідження є технології тренування нейронних мереж та їх оптимізації.

Практичне значення одержаних результатів – створення програмного забезпечення з підтримкою графічного інтерфейсу для тренування та оптимізації нейромережевої системи, яке дозволить спростити та пришвидшити процес створення моделей класифікації та регресії.

Інформаційну базу дослідження становлять наукові публікації та розробки зарубіжних вчених у областях машинного навчання, глибокого навчання, розробки інтерфейсу для десктопних застосунків, програмна документація використаних пакетів, офіційні матеріали, тощо.

Структура роботи складається зі вступу, чотирьох основних розділів з підрозділами, висновків, переліку використаних джерел та додатку програмного коду.

РОЗДІЛ 1
ПОСТАНОВКА ЗАВДАННЯ

1.1 Визначення вимог до роботи

Необхідно створити експертну систему для контролю, оптимізації та візуалізації процесу тренування експертних систем. Кінцевим продуктом має стати застосунок, що здатний тренувати та експортувати нейромережеві регресійні та класифікаторні моделі для подальшого використання в аналізі даних.

Актуальність теми.

У сучасному світі нейромережеві експертні системи знаходять дедалі ширше застосування в різних сферах, таких як медицина, фінанси та промисловість. Проте їх ефективність і надійність часто потребують додаткового контролю та оптимізації. Це підкреслює необхідність розробки спеціалізованого програмного забезпечення, яке забезпечить моніторинг та підвищення продуктивності цих систем.

Метою даної дипломної роботи є розробка програмного забезпечення, яке:

1. Дозволить здійснювати моніторинг продуктивності нейромережевих експертних систем.
2. Визначатиме ключові показники ефективності для оцінки їх роботи.
3. Впроваджуватиме алгоритми оптимізації для покращення навчання та адаптації системи.
4. Забезпечить інтерактивний інтерфейс для користувачів для доступу до аналітичних даних та рекомендацій.

Завдання дипломної роботи.

1. Провести аналіз існуючих методів контролю та оцінки ефективності нейромережевих систем.
2. Розробити модулі для моніторингу продуктивності, включаючи візуалізацію даних.
3. Впровадити алгоритми оптимізації для покращення швидкості навчання та точності прогнозів.
4. Створити користувацький інтерфейс для управління параметрами нейромережевих експертних систем.
5. Провести тестування розробленого програмного забезпечення на реальних даних для оцінки його ефективності.

Для розробки програмного забезпечення використати IDE Visual Studio Code, бібліотеку Scikit-Learn. У якості мови програмування використати Python.

РОЗДІЛ 2
АНАЛІЗ ПРОБЛЕМИ ДОСЛІДЖЕННЯ

2.1. Аналіз доступних нейромережевих технологій

Існує значна кількість фреймворків, що використовуються для розробки та тренування нейронних мереж. Ці інструменти надають різноманітні можливості для реалізації моделей глибокого навчання з урахуванням різних вимог і специфікацій. Вибір відповідного фреймворку залежить від конкретних завдань, ресурсів та характеристик обраної моделі. Найпопулярнішими опціями є наступні.

TensorFlow – це інтерфейс для вираження алгоритмів машинного навчання та реалізація для виконання таких алгоритмів. З допомогою TensorFlow, обчислення можуть бути виконані на різноманітних гетерогенних системах, починаючи від мобільних пристроїв, таких як телефони та планшети, і закінчуючи великомасштабними розподіленими системами з сотнями машин і тисячами обчислювальних пристроїв, таких як відеокарти.

Система Tensorflow є гнучкою і може бути використана для вираження широкого спектру алгоритмів, включаючи алгоритми навчання і виведення для моделей глибоких нейронних мереж, і використовується для проведення досліджень і розгортання систем машинного навчання у виробництві в більш ніж десяти областях інформатики та інших галузях, включаючи розпізнавання мови, комп’ютерний зір, робототехніку, пошук інформації, обробку природної мови та видобуток географічної інформації. API TensorFlow та еталонна реалізація були випущені як пакет з відкритим вихідним кодом під ліцензією Apache 2.0 у листопаді 2015 року. [7]

TF Lite – це пакет спеціального функціоналу бібліотеки, в основному призначений для застосування моделей на мобільних пристроях. За допомогою

TensorFlow Lite можна перетворити існуючі моделі в оптимізовану версію у вигляді файлу TF Lite.

TF надає різні методи для оптимізації, стиснення та перетворення моделі МН у формат TF Lite. Це полегшена версія, яка підтримує більшість функцій звичайної бібліотеки TF.

Наразі TF використовується у світі для різних застосувань. Він полегшує реалізацію алгоритмів МН і побудови прогнозів для додатків ШІ високої обчислювальної потужності. [8] TF Lite, з іншого боку, є платформою, в першу чергу, для дешевого обчислювального обладнання.

Окрім переваг TF Lite, пов’язаних із затримкою та розміром, фреймворк забезпечує безпеку даних, навчаючись локально на пристрої. Також відсутня вимога щодо доступу до Інтернету. Як результат, середовища застосування TF Lite не обмежуються локаціями з широкополосним покриттям. Моделі TF Lite зберігаються у файлі мостового формату у вигляді пласких буферів. Це інструментарій серіалізації, який зберігає дані в такому сплощеному необробленому буфері, забезпечуючи повний доступ до них перед пакуванням. Такий підхід до формату дозволяє оптимізувати обчислення і мінімізувати потребу в дисковій пам’яті.

Keras – це бібліотека з відкритим вихідним кодом, яка надає інтерфейс мовою програмування Python для створення штучних нейронних мереж. Keras діє як інтерфейс для бібліотеки TensorFlow.

Keras містить численні реалізації загальновживаних нейромережевих будівельних блоків, таких як шари, цілі, функції активації, оптимізатори, а також високу кількість інструментів для полегшення роботи з графічними та текстовими даними, щоб спростити кодування глибоких нейронних мереж. Код Keras доступний на платформі GitHub. [9]

На додаток до стандартних нейронних мереж, Keras підтримує згорткові та рекурентні нейронні мережі. Інтерфейс також підтримує інші поширені утиліти, такі як відсіювання, пакетна нормалізація та об’єднання. [7]

Keras дозволяє користувачам створювати глибокі моделі на смартфонах (iOS і Android), в Інтернеті або на віртуальній машині. Він також дозволяє використовувати розподілене навчання моделей глибокого навчання на кластерах графічних процесорів (GPU) і тензорних процесорів (TPU).

Scikit-learn — це одна з найвідоміших бібліотек Python для машинного навчання, що забезпечує набір ефективних та зручних у використанні інструментів для виконання широкого спектра задач аналізу даних. Її функціонал охоплює алгоритми класифікації, регресії, кластеризації, вибору ознак, зменшення розмірності, а також засоби для передобробки даних. Бібліотека створена на основі NumPy, SciPy та Matplotlib, що забезпечує її інтеграцію з іншими популярними інструментами Python для наукових обчислень.

Ключові особливості бібліотеки Scikit-learn

Scikit-learn підтримує широкий набір алгоритмів, які дозволяють вирішувати різноманітні задачі:

- класифікація: логістична регресія, дерева рішень, метод опорних векторів (SVM), ансамблеві методи (Random Forest, Gradient Boosting);
- регресія: лінійна та поліноміальна регресія, регресія на основі дерева рішень, Lasso, Ridge, ElasticNet;
- кластеризація: k-means, агломеративна кластеризація, DBSCAN;
- зменшення розмірності: метод головних компонент (PCA), t-SNE, ізомап.

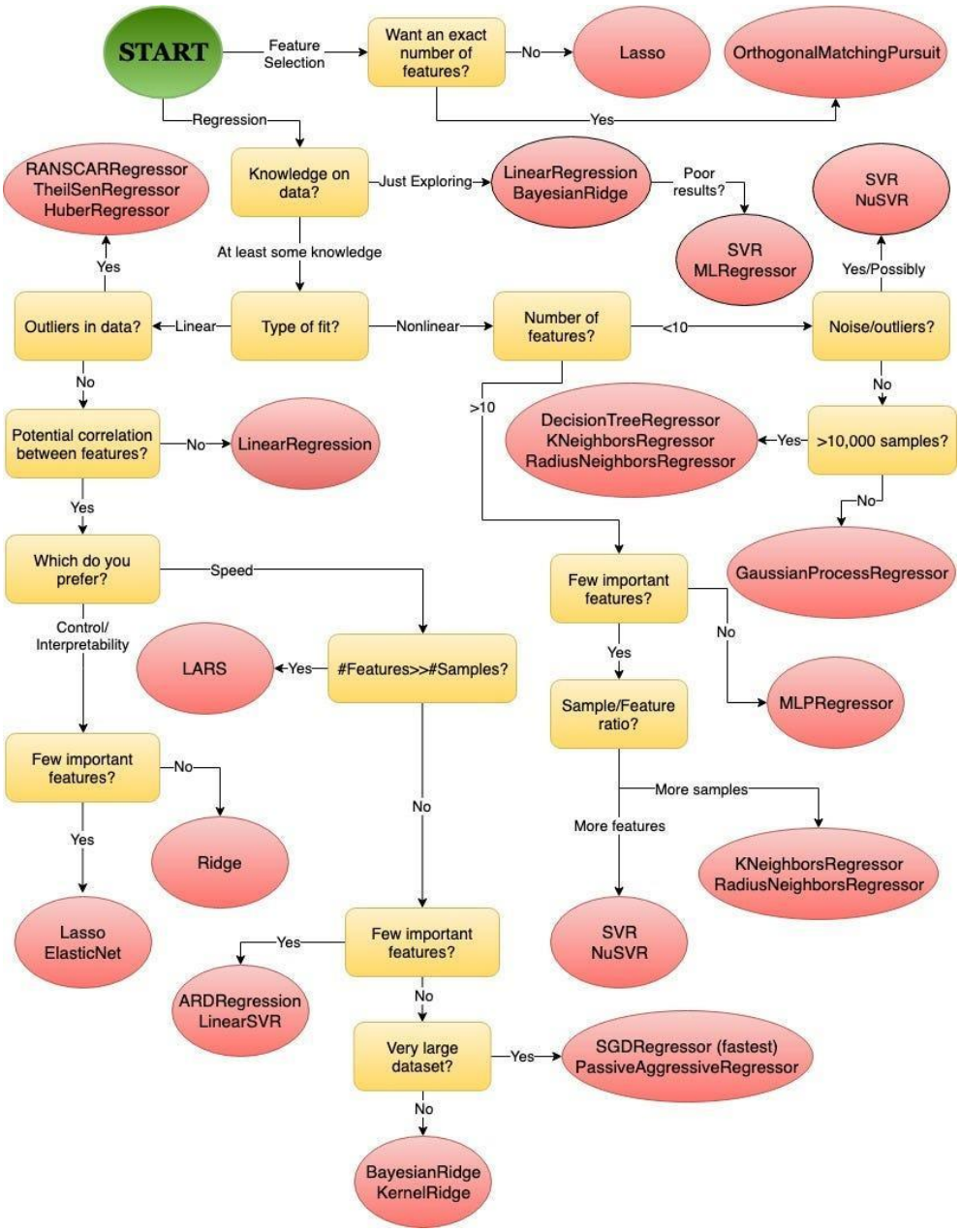


Рисунок 2.1 – Алгоритм вибору інструментарію Scikit-Learn

Бібліотека також пропонує обширні засоби для передобробки даних, їх підготовки до аналізу, включаючи:

- масштабування (стандартизація, нормалізація);
- обробку пропущених значень;
- кодування категоріальних змінних;
- фільтрацію та вибір ознак;
- перехресна перевірка та оцінка моделей.

Крім цього, Scikit-learn забезпечує розвинені інструменти для перевірки якості моделей:

- метрики (точність, F1-міра, середньоквадратична похибка, ROC-AUC тощо);
- методи крос-валідації;
- підбір гіперпараметрів через GridSearchCV та RandomizedSearchCV.

Бібліотека є модульною та підтримує конвеєрний підхід (Pipeline), що дозволяє зручно комбінувати етапи передобробки даних і побудови моделі. Це зменшує кількість коду і підвищує зручність роботи.

Завдяки відкритому коду бібліотека легко розширюється і може використовуватися спільно з іншими інструментами Python, такими як TensorFlow або PyTorch, для вирішення більш складних задач.

Усі ці переваги роблять її оптимальною для створення ПЗ контролю та оцінки нейронних систем.

Seaborn — це бібліотека Python, яка забезпечує засоби для створення високоякісних графіків із акцентом на статистичний аналіз даних. Вона розроблена на основі Matplotlib, але значно спрощує створення складних графічних побудов, автоматизуючи процес налаштування та інтегруючи статистичні аспекти візуалізації. Seaborn органічно працює з об’єктами Pandas DataFrame, що дозволяє безпосередньо аналізувати табличні дані та створювати інформативні графіки без потреби в додатковій підготовці.

Головною перевагою Seaborn є її орієнтація на простоту використання та можливість швидкого створення графіків, які відображають складні взаємозв’язки між змінними. Вона підтримує візуалізацію розподілів, зв’язків між змінними та категоріальних даних, забезпечуючи можливість автоматичного нанесення статистичних моделей, таких як регресійні лінії або довірчі інтервали. Це робить її зручним інструментом для дослідників, які працюють із даними соціального, економічного чи природничого характеру.

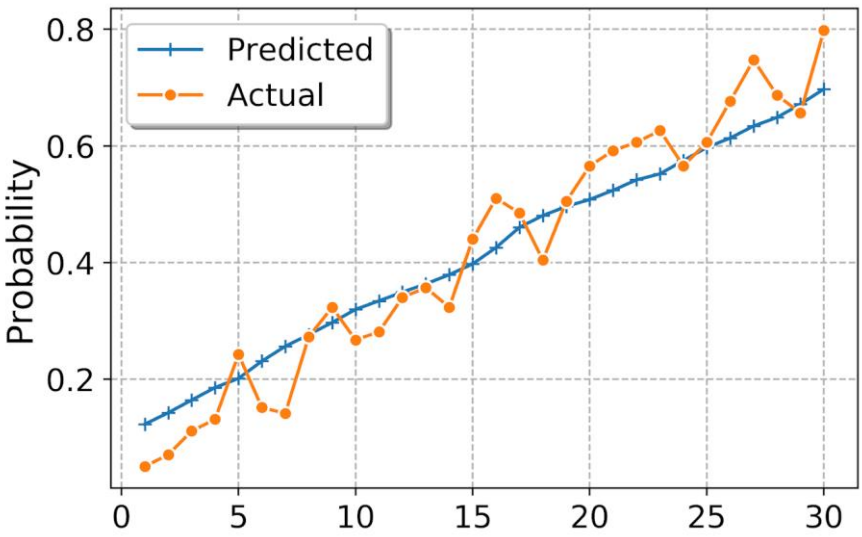


Рисунок 2.2 – Приклад графіку створеного за допомогою Seaborn

Крім того, Seaborn дозволяє гнучко налаштовувати зовнішній вигляд графіків, включаючи стиль, кольорові палітри та анотації. Завдяки цьому вона є універсальним інструментом, що поєднує потужність, естетичність і легкість використання.

Pandas: Потужний інструмент для обробки та аналізу даних

Pandas — це бібліотека Python, створена для роботи з даними в табличному форматі. Вона забезпечує зручні та високоефективні засоби для маніпуляцій, аналізу та візуалізації структурованих даних. Pandas є основним інструментом у сфері аналізу даних і використовується у наукових дослідженнях, машинному навчанні, фінансовій аналітиці та багатьох інших галузях.

```
In [34]: data.describe()
```

Out[34]:

	Area Code	Item Code	Element Code	latitude	longitude	Y1961	Y1962	Y19
count	21477.000000	21477.000000	21477.000000	21477.000000	21477.000000	17938.000000	17938.000000	17938.0000
mean	125.449411	2694.211529	5211.687154	20.450613	15.794445	195.262069	200.782250	205.4646
std	72.868149	148.973406	146.820079	24.628336	66.012104	1864.124336	1884.265591	1861.1747
min	1.000000	2511.000000	5142.000000	-40.900000	-172.100000	0.000000	0.000000	0.0000
25%	63.000000	2561.000000	5142.000000	6.430000	-11.780000	0.000000	0.000000	0.0000
50%	120.000000	2640.000000	5142.000000	20.590000	19.150000	1.000000	1.000000	1.0000
75%	188.000000	2782.000000	5142.000000	41.150000	46.870000	21.000000	22.000000	23.0000
max	276.000000	2961.000000	5521.000000	64.960000	179.410000	112227.000000	109130.000000	106356.0000

8 rows × 58 columns

Рисунок 2.3 – Приклад візуалізації таблиці з бібліотекою Pandas

Основою роботи з Pandas є два основних типи даних: Series (одновимірні структури, аналогічні спискам чи стовпцям таблиць) і DataFrame (двовимірні структури, які нагадують електронні таблиці або бази даних). Ці структури забезпечують інтуїтивний доступ до даних через індекси та метадані.

Pandas дозволяє виконувати широкий спектр операцій, таких як зчитування та збереження даних у різних форматах (CSV, Excel, SQL, JSON тощо), фільтрація, сортування, агрегація, групування, злиття таблиць та обчислення нових змінних. Її функціонал оптимізований для роботи з великими наборами даних, що робить бібліотеку швидкою та ефективною.

Однією з ключових переваг Pandas є інтеграція з іншими бібліотеками Python, такими як NumPy, Matplotlib та Seaborn, що дозволяє створювати цілісні аналітичні та візуалізаційні конвеєри. Завдяки цьому Pandas є незамінним інструментом для дослідників та розробників, які працюють із великими обсягами даних.

Pandas дозволяє легко обробляти пропущені дані, перетворювати типи змінних, виконувати часові операції та багато іншого. Її гнучкість і продуктивність зробили бібліотеку стандартом де-факто у сфері аналізу даних.

2.2. Аналіз доступних середовищ розробки

ML.NET

ML.NET — це безкоштовна бібліотека машинного навчання для мов програмування C# і F#. Вона також підтримує моделі Python у поєднанні з NimbusML. Попередня версія ML.NET містила трансформації для інженерії ознак, такі як створення n-грамів, а також навчальні алгоритми для розв’язання завдань бінарної класифікації, багатокласової класифікації та регресії. Додаткові задачі машинного навчання, такі як виявлення аномалій і системи рекомендацій, були додані пізніше, а інші підходи, такі як глибоке навчання, будуть включені в майбутніх версіях.

ML.NET надає аналітичні та прогностичні можливості, основані на моделях, існуючим розробникам .NET. Фреймворк побудовано на основі .NET Core та .NET Standard, що дозволяє йому працювати крос-платформно на Linux, Windows та macOS. Хоча фреймворк ML.NET новий, його корені беруть початок у 2002 році як проект Microsoft Research під назвою TMSN (текстове видобування, пошук і навігація), що використовувався всередині продуктів Microsoft. Пізніше його перейменували в TLC (кодування навчання) приблизно у 2011 році. ML.NET виник з бібліотеки TLC і значно перевищив свого «батька», за словами доктора Джеймса МакКефрі, представника Microsoft Research.

Розробники можуть навчати модель машинного навчання або повторно використовувати існуючу модель від третьої сторони та запускати її в будь-якому середовищі офлайн. Це означає, що розробникам не потрібно мати фон у галузі науки про дані для використання фреймворку. Підтримка формату моделі глибокого навчання Open Neural Network Exchange (ONNX) була впроваджена з версії 0.3 в ML.NET. Випуск також включав інші помітні вдосконалення, такі як Факторизаційні Машини, LightGBM, ансамблі, трансформацію LightLDA та OVA. Інтеграція ML.NET з TensorFlow була реалізована з випуску 0.5. Підтримка застосунків x86 і x64 була додана в версії 0.7, включаючи розширені можливості рекомендацій з використанням факторизації матриць. Повний план запланованих функцій доступний на офіційному репозиторії GitHub.

Перша стабільна версія 1.0 фреймворку була оголошена на конференції розробників Build 2019. Вона включала новий інструмент Model Builder і можливості AutoML (автоматизоване машинне навчання). Версія 1.3.1 представила попередній перегляд навчання глибоких нейронних мереж за допомогою C# зв’язків для TensorFlow та завантажувача бази даних, який дозволяє навчати моделі на базах даних. Попередній перегляд 1.4.0 додав оцінку ML.NET на процесорах ARM та навчання глибоких нейронних мереж з використанням GPU для Windows і Linux. [31]

У доповіді Microsoft про машинне навчання з використанням ML.NET

було продемонстровано, що він здатний навчати моделі для аналізу настроїв, використовуючи великі набори даних, досягаючи високої точності. Результати показали 95% точності на наборі даних рецензій Amazon обсягом 9 ГБ.

Model Builder — це командний інтерфейс (CLI), який використовує ML.NET AutoML для навчання моделей і вибору найкращого алгоритму для даних. Попередній перегляд ML.NET Model Builder є розширенням для Visual Studio, яке використовує ML.NET CLI та ML.NET AutoML для створення найкращої моделі ML.NET за допомогою графічного інтерфейсу.

Тема справедливості та пояснюваності ШІ стала предметом обговорення для етики в AI в останні роки. Основною проблемою для застосувань машинного навчання є ефект чорного ящика, коли кінцеві користувачі та розробники програми не знають, як алгоритм прийшов до рішення, чи містить набір даних упередження. Версія 0.8 включала API для пояснюваності моделей, які використовувалися всередині Microsoft. Це додало можливість зрозуміти важливість ознак моделей з додаванням «Загальної важливості ознак» та «Узагальнених адитивних моделей».

Коли є кілька змінних, які впливають на загальний бал, можна побачити розподіл кожної змінної та які ознаки мали найбільший вплив на остаточний бал. Офіційна документація демонструє, що метрики оцінювання можна виводити для налагодження. Під час навчання та налагодження моделі розробники можуть переглядати та перевіряти живі фільтровані дані, використовуючи інструменти DataView у Visual Studio.

Infer.NET — це популярний фреймворк для машинного навчання на основі моделей, який використовується для досліджень в академічних установах з 2008 року. Microsoft Research оголосила про його відкритий вихід, і тепер він є частиною фреймворку ML.NET. Infer.NET використовує ймовірнісне програмування для опису ймовірнісних моделей, що надає додаткову перевагу в інтерпретованості. Простір імен Infer.NET було змінено на Microsoft.ML.Probabilistic у відповідності з просторами імен ML.NET.

Microsoft визнала, що мова програмування Python популярна серед науковців даних, тому вона представила NimbusML — експериментальні прив'язки Python для ML.NET. Це дозволяє користувачам навчати та використовувати моделі машинного навчання в Python. Він також був відкритий, подібно до Infer.NET.

ML.NET дозволяє користувачам експортувати навчені моделі у формат ONNX. Це відкриває можливість використовувати моделі в різних середовищах, які не використовують ML.NET. Ці моделі можна запускати на стороні клієнта в браузері за допомогою ONNX.js — JavaScript-фреймворку для моделей глибокого навчання, створених у форматі ONNX.

Разом із запуском попередньої версії ML.NET Microsoft представила безкоштовні навчальні посібники та курси з ШІ, щоб допомогти розробникам зрозуміти техніки, необхідні для роботи з цим фреймворком. [26]

Google Colab.

Colab – це безкоштовне середовище для створення блокнотів на Jupyter, яке працює повністю в хмарі. Ключовою перевагою є те, що воно не потребує встановлення та налаштування. Colab підтримує багато популярних бібліотек машинного навчання. Для компіляції коду, Colab надає хмарне обладнання з 16 ГБ відеопам'яті, 11 ГБ оперативної пам'яті та 60 ГБ хранилища. Додатково, віртуальна машина має обмеження на 12 годин безперервної роботи.

Google Colab надає користувачам наступні можливості:

- редактор для написання та виконання коду на Python;
- документація коду, з підтримкою математичних рівнянь;
- створювати/завантажувати/поширювати блокноти;
- імпортувати/зберігати блокноти з/на Google Диск;
- імпортувати/публікувати блокноти з GitHub;
- імпорт зовнішніх наборів даних, наприклад, з Kaggle;
- інтеграція PyTorch, TensorFlow, Keras, OpenCV;
- безкоштовний хмарний сервіс з безкоштовним графічним

процесором.

Visual Studio Code: Середовище розробки для Python-застосунків

Visual Studio Code (VS Code) — це легке, але потужне кросплатформне інтегроване середовище розробки (IDE), яке широко використовується для створення Python-застосунків. Завдяки своїй модульній структурі, численним розширенням і зручному інтерфейсу, VS Code забезпечує розробникам інструменти для ефективного написання, тестування та налагодження Python-коду.

VS Code підтримує Python через офіційне розширення, яке додає повний набір функцій для роботи з цією мовою програмування. Воно забезпечує підсвічування синтаксису, автодоповнення коду, перевірку типів, статичний аналіз коду та інтеграцію з віртуальними середовищами Python. Завдяки інтеграції з такими інструментами, як Pylint, Flake8 або mypy, розробники можуть проводити аналіз коду безпосередньо в IDE, що значно підвищує якість програмного забезпечення.

Вбудований термінал VS Code дозволяє виконувати Python-скрипти, управляти залежностями через pip або conda і налаштовувати віртуальні середовища, не покидаючи середовища розробки. Система інтеграції з налагоджувачем надає розширені можливості для відстеження помилок, дозволяючи ставити точки зупинки, відслідковувати змінні та аналізувати виконання коду в реальному часі.

VS Code також забезпечує зручність для розробки багатофайлових проєктів. Завдяки підтримці Git розробники можуть працювати з системами контролю версій, а інструменти для управління середовищами (віртуальні середовища, Docker) дозволяють легко створювати та підтримувати Python-застосунки в різних конфігураціях.

Можливість встановлення розширень робить VS Code надзвичайно гнучким. Наприклад, розширення для Flask або Django додають підтримку розробки веб-застосунків, а інтеграція з Jupyter Notebook дозволяє ефективно

працювати з даними та виконувати наукові обчислення.

Visual Studio Code поєднує легкість, швидкість та потужність, що робить його оптимальним вибором для Python-розробників, незалежно від складності проєкту чи рівня кваліфікації програміста. [29]

Qt Designer

Qt Designer — це графічний інструмент розробки інтерфейсів користувача, що входить до складу бібліотеки Qt, яка широко використовується для створення кросплатформного програмного забезпечення. Qt Designer забезпечує візуальне середовище для проєктування графічних інтерфейсів (GUI) шляхом компонування та налаштування віджетів без необхідності написання коду вручну. Це дозволяє значно прискорити процес розробки програм із сучасними та функціональними інтерфейсами.

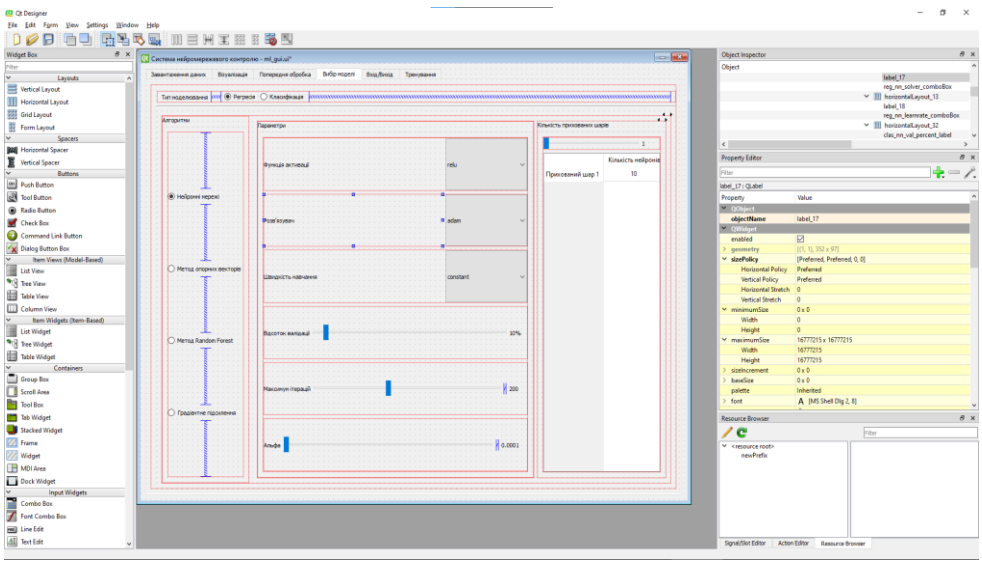


Рисунок 2.4 – Процес створення застосунку контролю експертних систем у програмному забезпеченні QT Designer

Основою роботи Qt Designer є використання концепції «WYSIWYG» (What You See Is What You Get), що забезпечує миттєвий перегляд майбутнього інтерфейсу. Всі створені макети зберігаються у форматі .ui, який є XML-файлом і може бути конвертований у програмний код за допомогою утиліт, таких як pyuic для Python.

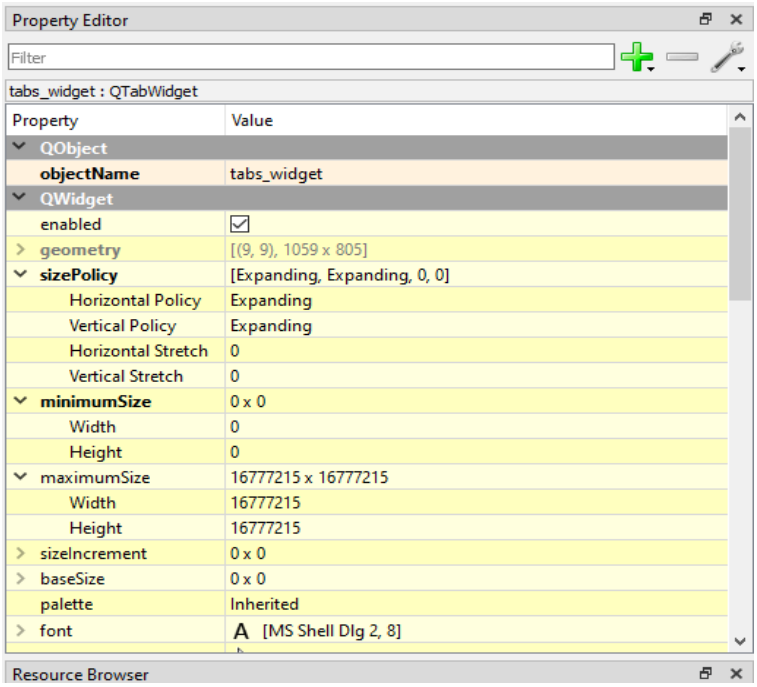


Рисунок 2.5 – Поля налаштування інтерфейсу QT Designer

Qt Designer підтримує широкий набір інструментів для розробки, включаючи інтерактивну роботу з віджетами, такими як кнопки, текстові поля, випадаючі списки та графічні області. Важливим аспектом є гнучке налаштування властивостей віджетів, їхньої поведінки та розташування за допомогою механізмів керування розкладкою. Це забезпечує адаптивність інтерфейсів до різних розмірів екранів та роздільних здатностей.

Бібліотека також дозволяє додавати сигнали та слоти, які є основою подієво-орієнтованої моделі Qt. Використовуючи цю функцію, розробник може налаштовувати інтерактивну поведінку елементів інтерфейсу без необхідності втручання у програмний код.

Qt Designer є універсальним інструментом, що підтримує інтеграцію з мовами програмування, такими як Python (через PyQt або PySide), C++ та іншими. Це робить його популярним серед розробників, які створюють складні інтерфейси для програмного забезпечення у різних галузях, від настільних додатків до вбудованих систем.

PyQt5 UI Code Generator — це інструмент, що входить до складу бібліотеки PyQt5 та використовується для автоматичного перетворення файлів інтерфейсу

користувача формату .ui, створених у Qt Designer, у програмний код на Python. Він спрощує процес інтеграції графічного інтерфейсу з додатком, усуваючи необхідність ручного написання коду для опису інтерфейсу.

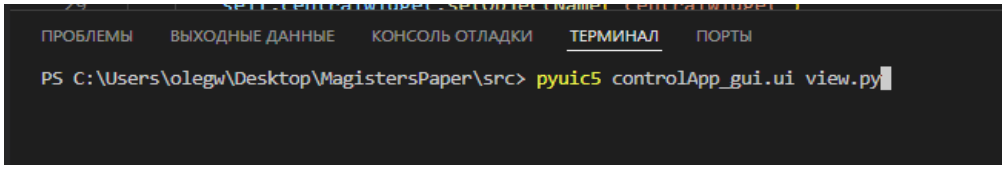


Рисунок 2.6 – Приклад роботи з бібліотекою PyQt5 у IDE VS Code

Утиліта працює шляхом трансформації XML-файлів, що описують структуру графічного інтерфейсу, у Python-модулі. Згенерований код містить класи, які відповідають за відображення, поведінку та властивості елементів інтерфейсу, включаючи віджети, розкладки, стилі, а також сигнали та слоти. Це дозволяє зберігати чіткий поділ між візуальним дизайном інтерфейсу та його логікою, що сприяє гнучкості та структурованості програмного забезпечення.

PyQt5 UI Code Generator надає можливість інтегрувати складні графічні інтерфейси у Python-додатки з мінімальними зусиллями. Генерація коду виконується через команду pyuic5, яка швидко обробляє вихідний файл і створює Python-скрипт, готовий до використання. Такий підхід не лише прискорює розробку, але й дозволяє розробникам зосередитися на бізнес-логіці програми, оскільки структура та візуальний вигляд інтерфейсу вже повністю визначені.

```
5  # Created by: PyQt5 UI code generator 5.15.9
6  #
7  # WARNING: Any manual changes made to this file will be lost when pyuic5 is
8  # run again.  Do not edit this file unless you know what you are doing.
9
10
11 from PyQt5 import QtCore, QtGui, QtWidgets
12
13
14 class Ui_MainWindow(object):
15     def setupUi(self, MainWindow):
16         MainWindow.setObjectName("MainWindow")
17         MainWindow.resize(1077, 823)
18         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)
19         sizePolicy.setHorizontalStretch(0)
20         sizePolicy.setVerticalStretch(0)
21         sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())
22         MainWindow.setSizePolicy(sizePolicy)
23         MainWindow.setMaximumSize(QtCore.QSize(16777215, 16777215))
24         icon = QtGui.QIcon()
25         icon.addPixmap(QtGui.QPixmap("../images/appIcon.png"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
26         MainWindow.setWindowIcon(icon)
27         MainWindow.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
28         self.centralwidget = QtWidgets.QWidget(MainWindow)
29         self.centralwidget.setObjectName("centralwidget")
```

Рисунок 2.7 – Конвертований Python-код створеного графічного інтерфейсу

Цей інструмент є незамінним для тих, хто створює програми з графічними інтерфейсами, забезпечуючи ефективну взаємодію між дизайнерськими та програмними етапами розробки. PyQt5 UI Code Generator є важливим компонентом у створенні Python-додатків із сучасними та функціональними графічними інтерфейсами. [30]

2.3. Проблематика навчання нейронної мережі

Висока кількість помилок у процесах контрольованого машинного навчання полягає у перепідгоні (перенавчанні). Модель погано узагальнює спостережувані дані на неспостережувані, що називається перенавчанням. Через існування перенавчання модель чудово працює на навчальній вибірці, але погано – на тестовій.

Це явище пов’язано з тим, що надмірно підігнана модель не може впоратися з фрагментами інформації на тестовій вибірці, що значно відрізняються від тих, що містились в навчальній вибірці. З іншого боку, надмірно підігнані моделі мають тенденцію запам’ятовувати всі дані, включаючи неминучий шум на навчальній вибірці, замість того, щоб вивчати об’єкт, прихований за цими даними.

Причини цього явища є складними. Загалом, їх можна розділити на три типи:

1) навчання з шумом на навчальній вибірці: коли навчальна вибірка занадто мала за розміром, або містить менш репрезентативні дані, або занадто багато шумів. У такій ситуації шуми мають великі шанси бути вивченими, а згодом стати основою для прогнозів. Отже, добре функціонуючий алгоритм повинен вміти відрізняти репрезентативні дані від шумів; [17]

2) складність гіпотези: компроміс у складності, ключове поняття в статистиці та машинному навчанні, – це компроміс між дисперсією та

упередженістю. Це стосується балансу між точністю та узгодженістю. Коли алгоритми мають занадто багато гіпотез (занадто багато вхідних даних), модель стає в середньому більш точною при меншій узгодженості. [17] Така ситуація означає, що моделі можуть кардинально відрізнятися на різних наборах даних;

3) процедури множинних порівнянь, які повсюдно використовуються в алгоритмах індукції, а також в інших алгоритмах штучного інтелекту. Під час цих процесів ми завжди порівнюємо кілька елементів на основі оцінок, отриманих за допомогою оціночної функції, і вибираємо елемент з максимальною оцінкою. Однак цей процес, ймовірно, обирає деякі елементи, які не покращують або навіть знижують точність класифікації.

Протилежністю надмірного навчання є недостатнє навчання. Недостатній підгон виникає, коли дані навчання ще можна було б покращити. Це може статися з кількох причин: Якщо модель недостатньо потужна, надмірно зарегульована або просто недостатньо довго навчалася. Це означає, що мережа не вивчила відповідні закономірності в навчальних даних.

Щоб запобігти надмірному пристосуванню, найкращим рішенням є використання більш повних навчальних даних. Набір даних повинен охоплювати весь діапазон вхідних даних, з якими має працювати модель. Додаткові дані можуть бути корисними лише тоді, коли вони охоплюють нові та цікаві випадки.

Модель, навчена на більш повних даних буде краще узагальнювати. Коли це вже неможливо, наступним найкращим рішенням є використання таких методів, як регуляризація. Вона накладає обмеження на кількість і тип інформації, яку може зберігати модель.

Якщо модель може дозволити собі запам’ятати лише невелику кількість шаблонів, процес оптимізації змусить її зосередитися на найпомітніших шаблонах, які мають більше шансів на хороше узагальнення. [8]

2.4. Аналіз існуючих способів оптимізації точності нейронної мережі

Рання зупинка.

Ця стратегія використовується, щоб уникнути явища уповільнення швидкості навчання. Ця проблема означає, що точність алгоритмів перестає покращуватися після певного моменту, або навіть погіршується через навчання з шумом. Цей метод був започаткований у 1970-х роках у контексті ітерації Ландвебера. [18]

Даний метод також широко використовується в ітеративних алгоритмах, особливо в нейронних мережах, починаючи з 1990-х років. Якщо модель продовжуватиме навчатися після точки сповільнення навчання, то помилка валідації зростатиме, тоді як помилка навчання помилка навчання буде продовжувати зменшуватися.

Зменшення мережі.

Навчання з шумом є однією з важливих причин перенавчання. Отже, зменшення шуму є одним з основних способів запобігання перенавчання. На основі цього, застосовується обрізання у вигляді зменшення розміру кінцевих класифікаторів у реляційному навчанні, особливо в навчанні на основі дерева рішень. Обрізання використовується для зменшення складності класифікації шляхом усунення менш значущих або нерелевантних даних і для запобігання перенавчання і підвищення точності класифікації.

Початкове обрізання та пост-обрізання – це два стандартні підходи, що використовуються для боротьби з шумом:

Алгоритми початкового обрізання працюють під час процесу навчання. Вони використовують критерії зупинки, щоб визначити, коли припинити додавання умов до правила або додавання правила до опису моделі, наприклад, обмеження довжини кодування на основі оцінки вартості кодування; тестування значущості на основі значних відмінностей між розподілом позитивних і негативних прикладів; критерій зупинки відсікання на основі заздалегідь

визначеного порогу.

Пост-обрізання розбиває навчальну множину на дві підмножини: зростаючу множину та множину обрізання. Порівняно з попереднім обрізанням, алгоритми пост-обрізання ігнорують проблеми перенавчання під час процесу навчання на зростаючій множині. Замість цього вони запобігають перенавчанню, видаляючи умови та правила, що утворились під час навчання моделі. Цей підхід набагато точніший, але менш ефективний. [19]

2.5. Аналіз процесу оцінки імовірних збитків при класифікації

Матриця втрат представляє ціни різних помилок класифікації і множиться на вектор оцінених імовірностей, у результаті отримуємо вектор оцінок втрат, і кожне спостереження приписують тому класу, у якого буде найменша оцінка для ціни помилки. Оскільки правильна класифікація не несе жодних втрат, на головній діагоналі матриці втрат завжди стоять нулі, щодо решти коефіцієнтів матриці, то коефіцієнт, який стоїть на перетині n-го стовпчика та m-го рядка, є ціна неправильної класифікації спостереження, яке насправді належить до класу номер n, як такого, що належить до класу номер m.

Нейропакет дає змогу інтерпретувати відповідь навченої мережі як рівнем довіри, так і мірою помилки. Відмінності будуть лише у співвідношенні порогів ухвалення рішення і його відкидання. У другому випадку поріг прийняття рішення буде нижчим за поріг відкидання.

Мережа для матриці втрат має числові вхідні змінні та одну номінальну вихідну змінну, що відповідає виходу мережі, до якої додається матриця втрат. До набору даних додається також кілька числових змінних, що відповідають входам лінійної мережі.

Потім створюється лінійна мережа і вибираються ці змінні як незалежні, а потрібна вихідна змінна – як залежна. Далі встановлюються нульові значення порогів вихідного шару, і коефіцієнти матриці втрат поміщаються на місце всіх

інших ваг.

Після того як матрицю втрат побудовано, її додають до навченої мережі, що оцінює ймовірності, і в результаті отримують складову мережу, яка оцінює очікуваний збиток від ухваленого рішення.

У найпростішому варіанті оцінки ймовірності використовуються безпосередньо – спостереження належить до найімовірнішого класу.

Однак буває так, що одні помилки обходяться дорожче за інші. Наприклад, під час діагностики захворювання з можливим летальним результатом, призначення лікування людині, яка насправді здорова, можна вважати менш серйозною помилкою, ніж не призначення лікування хворому.

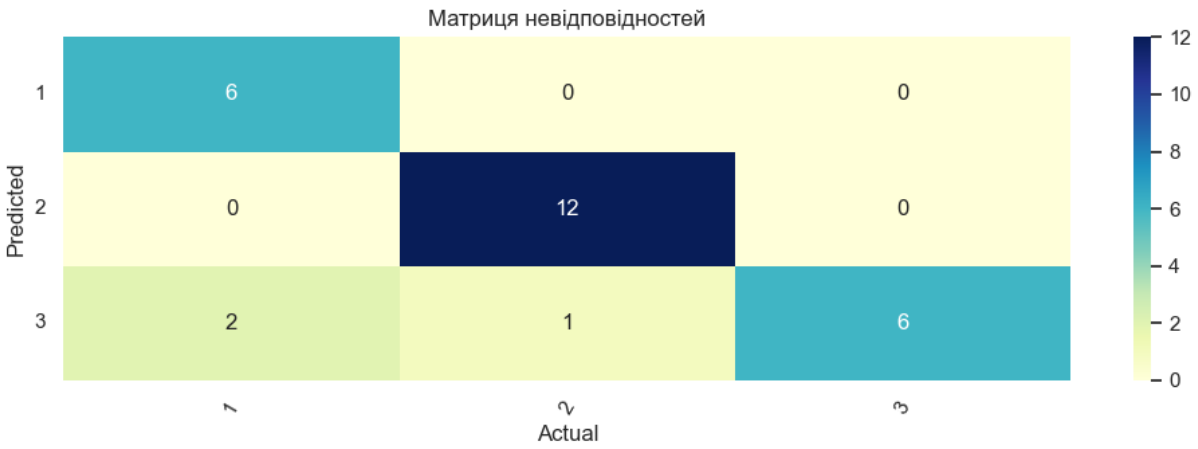


Рисунок 2.8 – Типовий вигляд графічного подання матриці втрат

Матриця втрат являє собою квадратну матрицю, коефіцієнтами якої є відносні ціни різних помилок класифікації. Вона множиться на вектор оцінених імовірностей, у результаті виходить вектор оцінок втрат, і кожне спостереження приписується тому класу, у якого буде найменша оцінка для ціни помилки.

Оскільки правильна класифікація не несе жодних втрат, на головній діагоналі матриці втрат завжди стоять нулі; щодо решти коефіцієнтів матриці, то коефіцієнт, який стоїть на перетині n-го стовпчика та m-го рядка, є ціна неправильної класифікації спостереження, яке насправді належить до класу номер n, як таке, що належить до класу номер m. [20]

2.6. Аналіз функцій активації нейронів

У процесі навчання нейромережевої моделі створюються шари згорткові шари з функцією активації “ReLU” та щільний вихідний шар з активацією “Softmax”.

Зрізаний лінійний вузол (rectified linear unit, ReLU) – це функція активації, яка вводить властивість нелінійності в модель глибокого навчання і вирішує проблему зникаючих градієнтів. Вона інтерпретує додатну частину свого аргументу. Це одна з найпопулярніших функцій активації в глибокому навчанні.

Функція ReLU математично може бути виражена як:

$$f(x) = x^+ = max(0, x) \tag{3.1}$$

Де x – це вхідне значення нейрона.

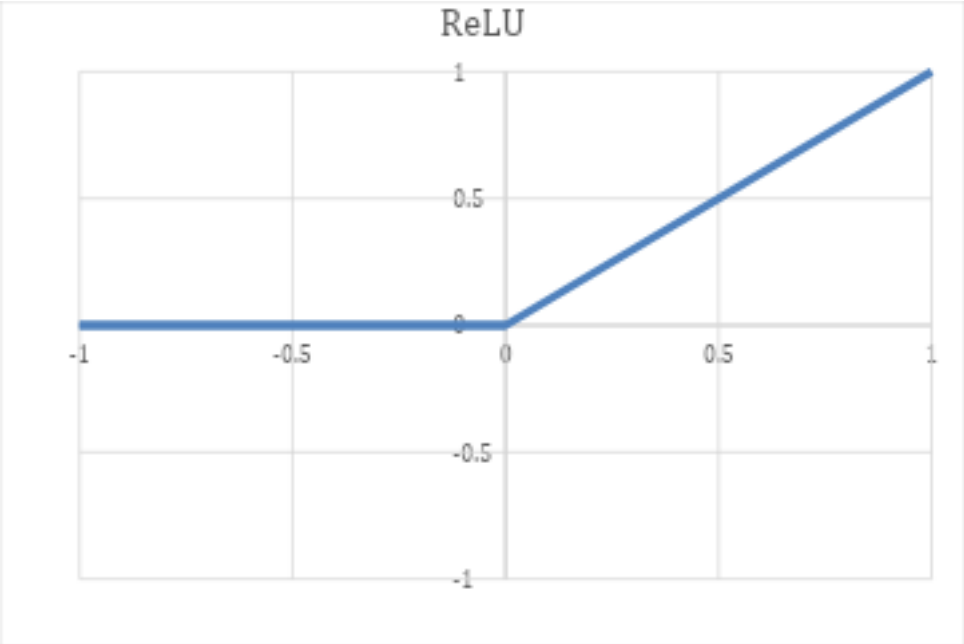


Рисунок 2.9 – Функція ReLU

Функція Softmax (укр. нормована експоненційна функція) – це узагальнення логістичної функції. Softmax застосовується в машинному навчанні для задач класифікації, коли кількість можливих класів більша за два (для двох класів використовується сігмоїдальна логістична функція).

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad (3.2)$$

Де $\vec{z}$ – вхідний вектор, e^{z_i}– експоненційна функція вхідного вектора, K – кількість класів в мультикласовому класифікаторі, e^{z_j} – експоненційна функція вихідного вектора.

Функція Identity (лінійна функція активації) у нейронних мережах визначається формулою:

$$f(x) = x \quad (3.3)$$

Це означає, що вихід нейрону безпосередньо дорівнює його вхідному значенню. На графіку ця функція представлена прямою лінією з нахилом 45° (коефіцієнт нахилу дорівнює 1).

Переваги

1. Простота реалізації: Лінійна функція не потребує складних обчислень, що робить її вкрай простою у використанні.

2. Інтерпретація вихідних даних: Оскільки вихід є пропорційним до входу, його легко інтерпретувати, особливо у регресійних задачах, де вихідні значення мають бути числовими.

3. Відсутність ефекту насичення: Identity-функція не має насичення, як у сигмоїди чи тангенса, що дозволяє уникнути проблеми «зникаючих градієнтів» у межах окремого нейрону.

Недоліки

1. Відсутність нелінійності: Identity є лінійною функцією, і якщо вона використовується у прихованих шарах нейронної мережі, це перетворює всю модель на лінійну. Це обмежує здатність мережі вивчати складні залежності в даних, оскільки композиція лінійних функцій є лінійною.

2. Погана диференціація: У глибоких мережах градієнти, обчислені через лінійну активацію, не стимулюють навчання через відсутність складної трансформації сигналу.

3. Не підходить для класифікації: Identity не обмежує вихід у межах

певного діапазону, що робить її непридатною для задач класифікації, де потрібні ймовірності або чітко визначені класи.

Випадки застосування

1. Регресійні задачі: Identity-функція часто використовується у вихідному шарі нейронних мереж для регресії, коли необхідно прогнозувати реальні числові значення (наприклад, ціну, температуру, час).

2. Прості моделі з лінійними залежностями: У випадках, коли дані мають переважно лінійний характер, і використання нелінійних функцій активації не дає значної переваги.

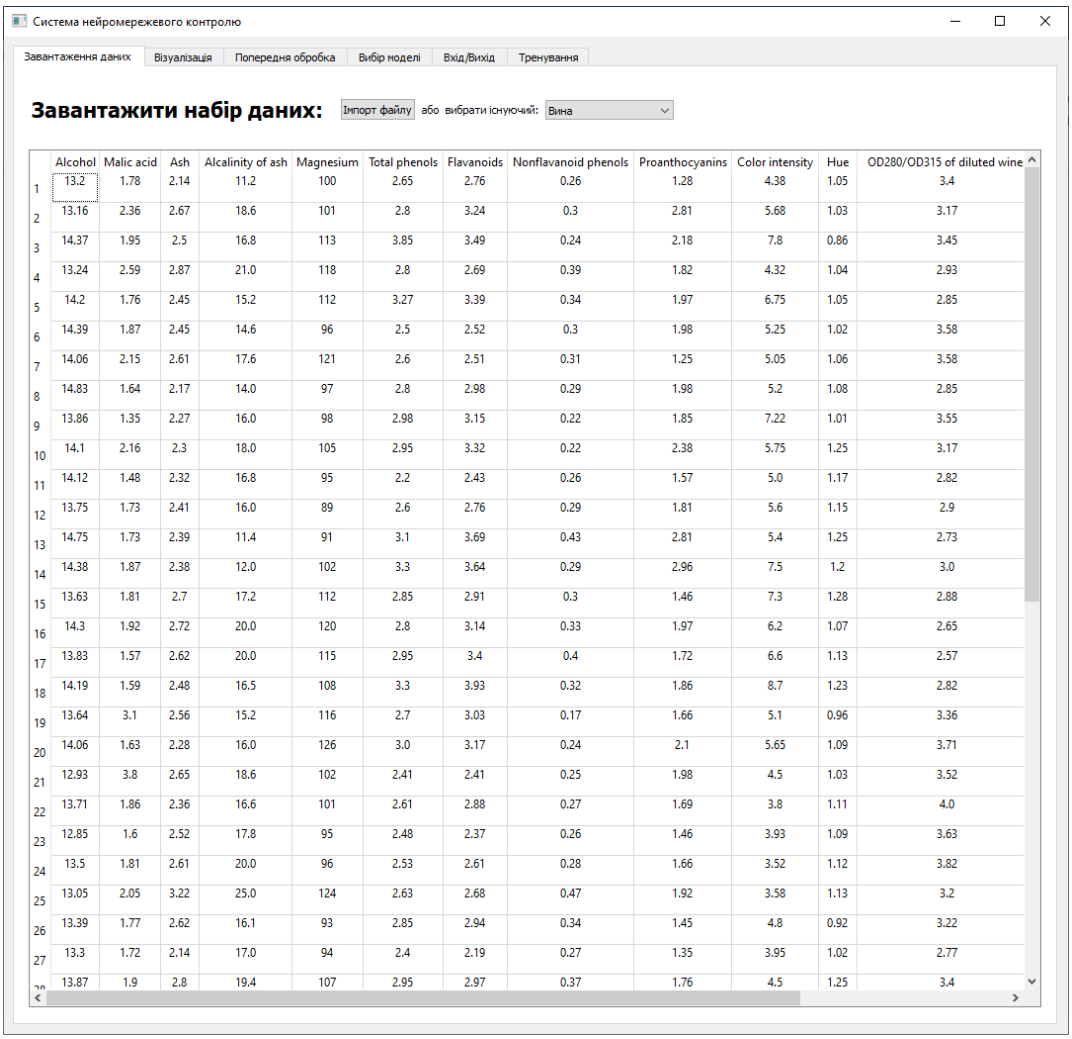
3. Поєднання з нелінійними шарами: У деяких архітектурах нейронних мереж лінійна функція може використовуватись у комбінації з нелінійними шарами для забезпечення специфічної трансформації сигналу.

Функція активації Identity є корисною у вузьких випадках, особливо у регресійних задачах або для побудови найпростіших моделей. Проте її лінійність обмежує її ефективність у прихованих шарах нейронних мереж, які повинні вивчати складні нелінійні залежності. У більшості сучасних задач машинного навчання її використання поступається місцем нелінійним функціям, таким як ReLU, сигмоїда або гіперболічний тангенс.

РОЗДІЛ 3
ОПИС ПОБУДОВИ СИСТЕМИ КОНТРОЛЮ НЕЙРОМЕРЕЖЕВОЇ КЛАСИФІКАЦІЇ

3.1. Побудова панелі завантаження даних

При запуску програми, користувач потрапляє на панель завантаження набору даних. Ця панель дозволяє імпортувати файл для подальшої обробки та переглянути його вміст.



	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wine
1	13.2	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.4
2	13.16	2.36	2.67	18.6	101	2.8	3.24	0.3	2.81	5.68	1.03	3.17
3	14.37	1.95	2.5	16.8	113	3.85	3.49	0.24	2.18	7.8	0.86	3.45
4	13.24	2.59	2.87	21.0	118	2.8	2.69	0.39	1.82	4.32	1.04	2.93
5	14.2	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85
6	14.39	1.87	2.45	14.6	96	2.5	2.52	0.3	1.98	5.25	1.02	3.58
7	14.06	2.15	2.61	17.6	121	2.6	2.51	0.31	1.25	5.05	1.06	3.58
8	14.83	1.64	2.17	14.0	97	2.8	2.98	0.29	1.98	5.2	1.08	2.85
9	13.86	1.35	2.27	16.0	98	2.98	3.15	0.22	1.85	7.22	1.01	3.55
10	14.1	2.16	2.3	18.0	105	2.95	3.32	0.22	2.38	5.75	1.25	3.17
11	14.12	1.48	2.32	16.8	95	2.2	2.43	0.26	1.57	5.0	1.17	2.82
12	13.75	1.73	2.41	16.0	89	2.6	2.76	0.29	1.81	5.6	1.15	2.9
13	14.75	1.73	2.39	11.4	91	3.1	3.69	0.43	2.81	5.4	1.25	2.73
14	14.38	1.87	2.38	12.0	102	3.3	3.64	0.29	2.96	7.5	1.2	3.0
15	13.63	1.81	2.7	17.2	112	2.85	2.91	0.3	1.46	7.3	1.28	2.88
16	14.3	1.92	2.72	20.0	120	2.8	3.14	0.33	1.97	6.2	1.07	2.65
17	13.83	1.57	2.62	20.0	115	2.95	3.4	0.4	1.72	6.6	1.13	2.57
18	14.19	1.59	2.48	16.5	108	3.3	3.93	0.32	1.86	8.7	1.23	2.82
19	13.64	3.1	2.56	15.2	116	2.7	3.03	0.17	1.66	5.1	0.96	3.36
20	14.06	1.63	2.28	16.0	126	3.0	3.17	0.24	2.1	5.65	1.09	3.71
21	12.93	3.8	2.65	18.6	102	2.41	2.41	0.25	1.98	4.5	1.03	3.52
22	13.71	1.86	2.36	16.6	101	2.61	2.88	0.27	1.69	3.8	1.11	4.0
23	12.85	1.6	2.52	17.8	95	2.48	2.37	0.26	1.46	3.93	1.09	3.63
24	13.5	1.81	2.61	20.0	96	2.53	2.61	0.28	1.66	3.52	1.12	3.82
25	13.05	2.05	3.22	25.0	124	2.63	2.68	0.47	1.92	3.58	1.13	3.2
26	13.39	1.77	2.62	16.1	93	2.85	2.94	0.34	1.45	4.8	0.92	3.22
27	13.3	1.72	2.14	17.0	94	2.4	2.19	0.27	1.35	3.95	1.02	2.77
28	13.87	1.9	2.8	19.4	107	2.95	2.97	0.37	1.76	4.5	1.25	3.4

Рисунок 3.1 – Панель завантаження даних

Кнопка «Імпорт файлу» (рис. 3.2) дає можливість завантажити файл з файлової системи пристрою. Програма підтримує популярні типи файлів, такі як «.csv» та «.xlsx»

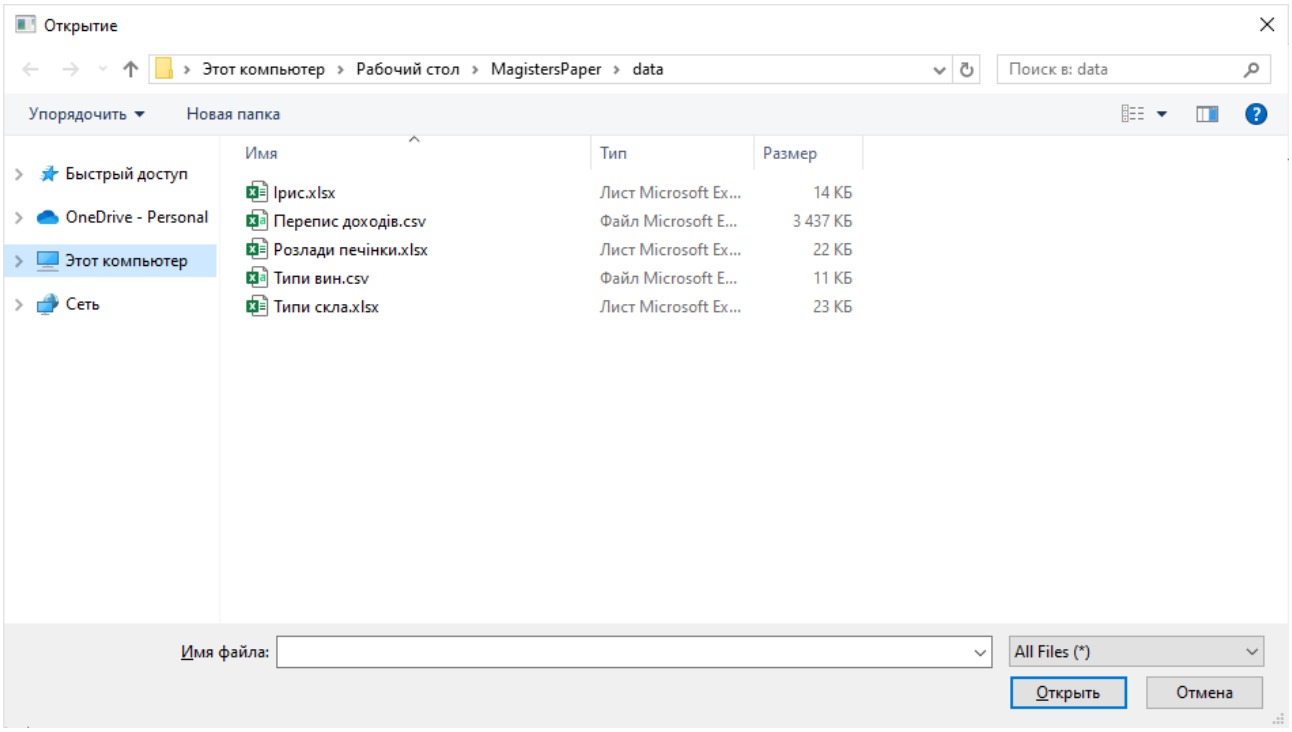


Рисунок 3.2 – Діалог завантаження файлу

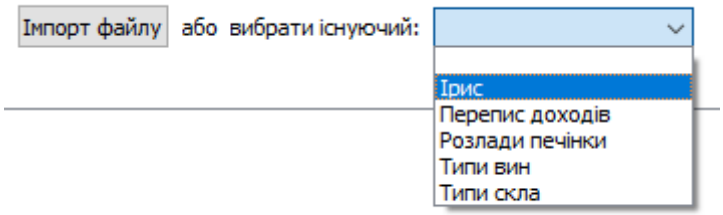


Рисунок 3.3 – Список вибору попередніх наборів даних

Для демонстрації процесу користування застосунком та його тестування, було завантажено декілька наборів даних з онлайн-ресурсу «UC Irvine Machine Learning Repository». [27]

3.2. Опис набору даних «Ірис»

Набір даних «Ірис» («Iris») є одним із найпопулярніших та найперших наборів даних, що використовуються для демонстрації методів класифікації в статистиці та машинному навчанні. Він був представлений знаменитим статистиком і біологом Рональдом Фішером у 1936 році і з того часу став стандартом для навчання алгоритмів класифікації. [28]

Структура набору даних: набір даних містить 150 екземплярів, поділених на три класи, по 50 екземплярів у кожному класі:

1. Iris-setosa.
2. Iris-versicolor.
3. Iris-virginica.

Кожен з класів відповідає одному з видів рослин ірису. Два з трьох класів (Iris-setosa та Iris-versicolor) лінійно відокремлюються один від одного, тоді як між класами Iris-versicolor та Iris-virginica існує певне перекриття, що ускладнює задачу класифікації. Один клас лінійно відокремлюється від двох інших; останні не є лінійно відокремленими один від одного. Це надзвичайно проста область.

Атрибути: Набір містить чотири числові ознаки, які описують фізичні характеристики кожної рослини:

1. Довжина чашолистка (sepal length).
2. Ширина чашолистка (sepal width).
3. Довжина пелюстки (petal length).
4. Ширина пелюстки (petal width).

Ці ознаки вимірюються в сантиметрах і є ключовими для класифікації виду ірису.

Прогнозована змінна: Прогнозованою змінною є клас рослини (один з трьох видів ірису). Типовою метою використання набору даних є класифікація ірису за його фізичними ознаками.

Особливості набору даних:

Набір даних має дуже просту структуру та є чудовим прикладом для початкових практик у машинному навчанні, зокрема для задач класифікації.

Кожен клас у наборі даних є чітко визначеним, і між класами існує певна розмежованість, що робить його ідеальним для застосування базових методів класифікації, таких як логістична регресія, KNN, дерева рішень тощо.

Важливою особливістю є те, що один клас Iris-setosa є лінійно відокремленим від двох інших, що дає змогу використовувати прості лінійні моделі для класифікації. Натомість для класів Iris-versicolor та Iris-virginica відокремлення вже є більш складним і потребує застосування більш складних алгоритмів.

Помилки в даних: У деяких екземплярах набору даних можуть бути незначні помилки, зокрема у значеннях деяких ознак. Наприклад:

- 35-й зразок має бути: 4.9, 3.1, 1.5, 0.2, «Iris-setosa», однак є помилка в четвертій ознаці;
- 38-й зразок має бути: 4.9, 3.6, 1.4, 0.1, «Iris-setosa», але містить помилки в другій та третій ознаках.

Застосування: Набір даних «Iris» широко використовується для навчання та тестування алгоритмів класифікації, а також для вивчення основних методів аналізу даних. Завдяки своїй простоті і водночас можливості для вивчення складніших моделей класифікації, цей набір даних є класичним прикладом для початкових курсів з машинного навчання та статистики.

3.3. Опис набору даних «Перепис доходів»

Набір даних «Adult Income» містить інформацію, яка дозволяє вивчати, як різні фактори впливають на річний дохід особи. Дохід залежить від багатьох змінних, серед яких рівень освіти, вік, стать, професія та інші демографічні та соціально-економічні характеристики. Це один із широко цитованих наборів

даних у контексті задач класифікації за допомогою методу K-найближчих сусідів (KNN) і є популярним прикладом для початкових практик в обробці даних та машинному навчанні.

Цільова змінна набору - Дохід: Ця змінна ділиться на два класи:

- <=50K — дохід менше або рівний 50 тисяч доларів на рік;
- 50K — дохід більший за 50 тисяч доларів на рік.

Атрибути (характеристики особи): У наборі даних 14 атрибутів, що описують демографічні та інші ознаки особи, такі як:

- вік;
- стать;
- рівень освіти;
- статус шлюбу;
- професія;
- тип працевлаштування;
- розмір родини;
- місце проживання;
- країна народження;
- дохід від пенсійного забезпечення;
- володіння акціями;
- стаж праці;
- тип професії;
- рівень освіти.

Метою аналізу цього набору даних є передбачення рівня доходу особи, виходячи з її особистих характеристик. Цей набір даних є корисним початковим прикладом для вивчення та практики обробки даних, виконання попередньої обробки даних та застосування алгоритмів машинного навчання для задачі класифікації.

Завдяки різноманітним атрибутам, цей набір даних дозволяє вивчати вплив соціально-економічних і демографічних факторів на рівень доходу. Це також

може бути корисним для розробки моделей, які дозволяють передбачити дохід людей на основі їхніх характеристик.

3.4. Опис набору даних «Типи скла»

Набір даних «Glass Classification» є частиною відомого репозиторію UCI Machine Learning Repository і містить інформацію, що використовується для класифікації різних типів скла. Зокрема, цей набір даних застосовується для задачі класифікації, де на основі фізичних та хімічних характеристик скла визначається його тип.

Структура набору даних:

Набір містить 214 записів (екземплярів), кожен з яких представляє зразок скла з певними властивостями. Атрибути (характеристики): Набір даних включає 10 атрибутів, що описують фізичні та хімічні характеристики скла:

- Id — унікальний ідентифікатор зразка скла;
- RI (Refractive Index) — показник заломлення скла;
- Na (Sodium) — вміст натрію в складі скла;
- Mg (Magnesium) — вміст магнію в складі;
- Al (Aluminum) — вміст алюмінію;
- Si (Silicon) — вміст кремнію;
- K (Potassium) — вміст калію;
- Ca (Calcium) — вміст кальцію;
- Ba (Barium) — вміст барію;
- Fe (Iron) — вміст заліза.

Цільова змінна: Type — тип скла, ціле число з 7 можливих значень, що вказують на різні типи скла. Ці типи можуть включати в себе різні варіанти скла, такі як «натрієве», «кремнеземне», «ванадієве» тощо.

Цей набір даних застосовується для розв’язання задач класифікації, де необхідно на основі характеристик скла (показник заломлення, вміст різних

елементів тощо) визначити тип скла. Він є корисним для тестування та порівняння різних алгоритмів класифікації, таких як методи підтримки векторів (SVM), дерева рішень, нейронні мережі, k-найближчих сусідів та інші. Набір даних «Glass Classification» є класичним прикладом для задач класифікації в машинному навчанні та статистиці.

3.5. Опис набору даних «Типи вин»

Набір даних «Wine Classification» містить результати хімічного аналізу вин, вирощених в одній області Італії, але отриманих з трьох різних сортів винограду. Цей набір даних широко використовується для задач класифікації, оскільки метою є визначення типу вина на основі кількості різних хімічних складових, знайдених у кожному з трьох типів вин.

Набір містить 178 записів, кожен з яких описує хімічний склад вина. Для характеризації складу вина у записах, набір даних включає 13 хімічних ознак:

1. Alcohol — вміст алкоголю в вині (відсотки).
2. Malic acid — вміст яблучної кислоти.
3. Ash — вміст попелу.
4. Alcalinity of ash — лужність попелу.
5. Magnesium — вміст магнію.
6. Total phenols — загальний вміст фенольних сполук.
7. Flavanoids — вміст флавоноїдів.
8. Nonflavanoid phenols — вміст нефлавоноїдних фенольних сполук.
9. Proanthocyanins — вміст проантоціанінів.
10. Color intensity — інтенсивність кольору вина.
11. Hue — відтінок вина.
12. OD280/OD315 of diluted wines — відношення оптичної густини вина при різних довжинах хвиль.
13. Proline — вміст проліну (амінокислоти).

Цільова змінна:

Class — тип вина, який позначений цілим числом, що відповідає одному з трьох класів, де кожен клас відповідає одному з трьох сортів винограду, використаних для виготовлення вина.

Цей набір даних є класичним прикладом для задач класифікації. Метою є класифікація вина на основі його хімічного складу. Кожен з трьох типів вина має свої характерні хімічні властивості, які дозволяють точно класифікувати його, що робить цей набір даних відмінним для тестування нових алгоритмів класифікації.

Особливості набору даних:

- набір даних має добре організовану структуру з чітко визначеними класами, що робить його корисним для перевірки алгоритмів класифікації, зокрема для початкового навчання на реальних даних;
- усі 13 ознак мають фізичне або хімічне значення, і кожна з них є важливою для визначення типу вина, що дозволяє добре виділити структуру класів.

Набір даних «Wine Classification» є ідеальним для тестування алгоритмів машинного навчання, зокрема таких, як методи підтримки векторів (SVM), логістична регресія, дерева рішень та інші методи класифікації. Завдяки своїй простоті та структурованості, він добре підходить для початкового тестування нових моделей, хоча не є надзвичайно складним для розв’язання.

3.6. Опис набору даних «Розлади печінки»

Набір даних «Liver Disorders» містить інформацію про медичні показники пацієнтів, що дозволяє проводити аналіз ймовірності захворювань печінки, зокрема тих, що можуть виникнути внаслідок надмірного споживання алкоголю. Всі дані в наборі належать чоловікам, і кожен запис у наборі представляє медичні результати одного індивіда.

Структура набору даних:

- кількість екземплярів: Набір містить 345 записів, кожен з яких відображає результати обстеження одного чоловіка.
- атрибути (характеристики): Набір містить 7 змінних:

1. Місячний рівень аланінамінотрансферази (ALT) — результат аналізу, що вимірює рівень ензиму в крові.
2. Місячний рівень аспартатамінотрансферази (AST) — показник іншого ферменту, який вказує на пошкодження печінки.
3. Гамма-глутамілтрансфераза (GGT) — ще один фермент, рівень якого підвищується при захворюваннях печінки.
4. Сума лужної фосфатази (ALP) — ще один показник функціонування печінки.
5. Білірубін — рівень білірубіну, який підвищується при порушеннях роботи печінки.
6. Кількість алкоголю, що споживається за день (Drinks) — вказує кількість алкогольних напоїв, спожитих на день (цифрове значення).
7. Selector (7-й атрибут) — змінна, яка була створена для поділу даних на тренувальну та тестову вибірки. Цей атрибут не є залежною змінною, що вказує на наявність або відсутність захворювання, і його не слід використовувати для класифікації.

Набір даних не містить явної змінної, що позначає наявність чи відсутність захворювання печінки. Однак, дослідники, які бажають використовувати цей набір даних для класифікаційних задач, можуть використовувати 6-й атрибут (споживання алкоголю) після його двійкового перетворення (наприклад, в значення «високе/низьке споживання») як залежну змінну для класифікації, вивчаючи вплив алкоголю на розвиток захворювань печінки.

Особливості набору даних:

- необхідність обережності при інтерпретації: 7-й атрибут, що часто неправильно інтерпретується як змінна для визначення наявності чи відсутності

захворювання печінки, не є такою змінною. Він був призначений лише для поділу даних на тренувальну та тестову вибірки, а не для використання в якості залежної змінної для класифікації;

- набір даних використовується для вивчення впливу алкоголю на розвиток порушень печінки, і для цієї мети дослідники можуть застосовувати методи двійкової класифікації, використовуючи змінну споживання алкоголю.

3.7. Побудова панелі візуалізації даних

Другою вкладкою є панель візуалізації даних. Ця панель відповідає за графічне відображення завантаженого набору даних та виводу статистики.

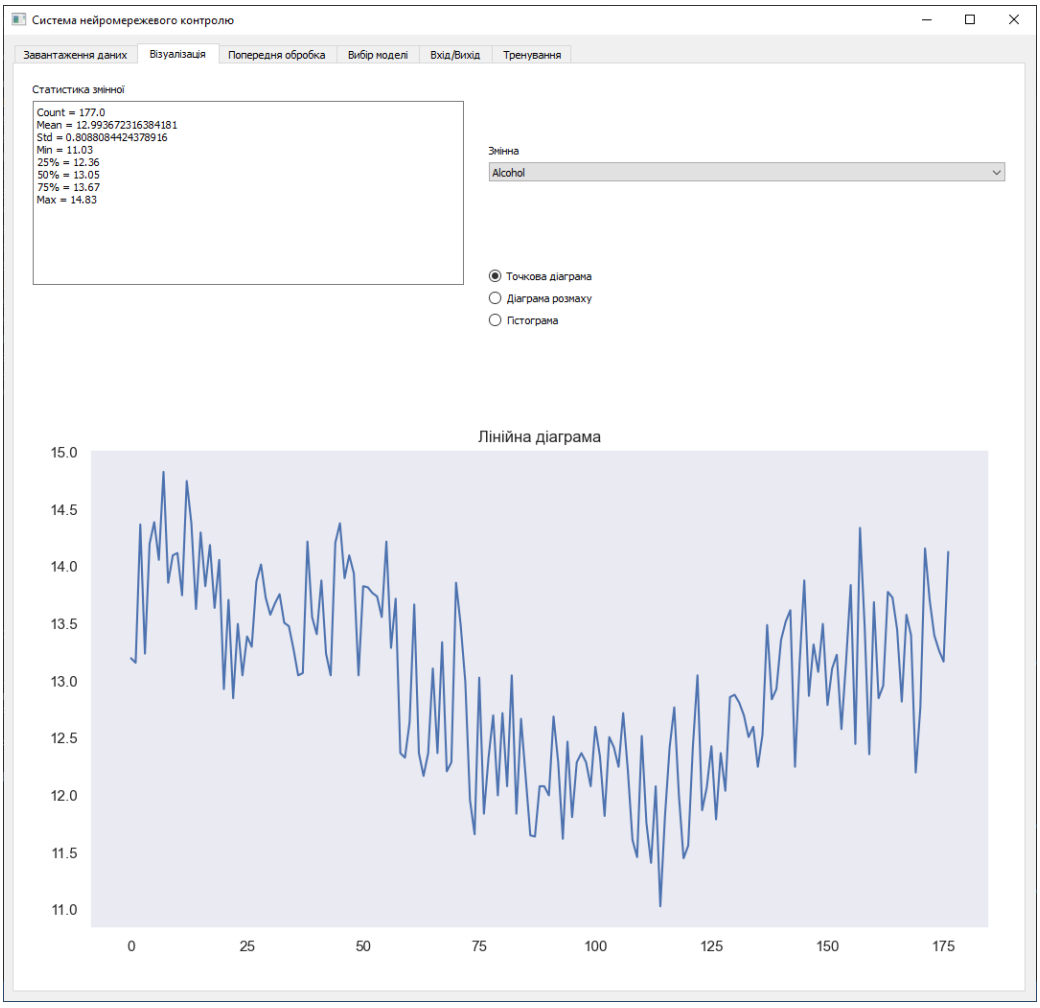


Рисунок 3.4 – Панель візуалізації даних

На цій панелі користувач може обрати одну зі змінних набору даних та побачити такі чисельні дані як кількість записів, середнє значення, середньоквадратичне відхилення, мінімальні та максимальне значення змінної (рис. 3.4), а також інтерпольований графік змінної за записами. Графік змінної можна переглянути у вигляді лінійної діаграми, діаграми розмаху або гістограми (рис. 3.5), залежно від найбільш доцільного вигляду подання даних.

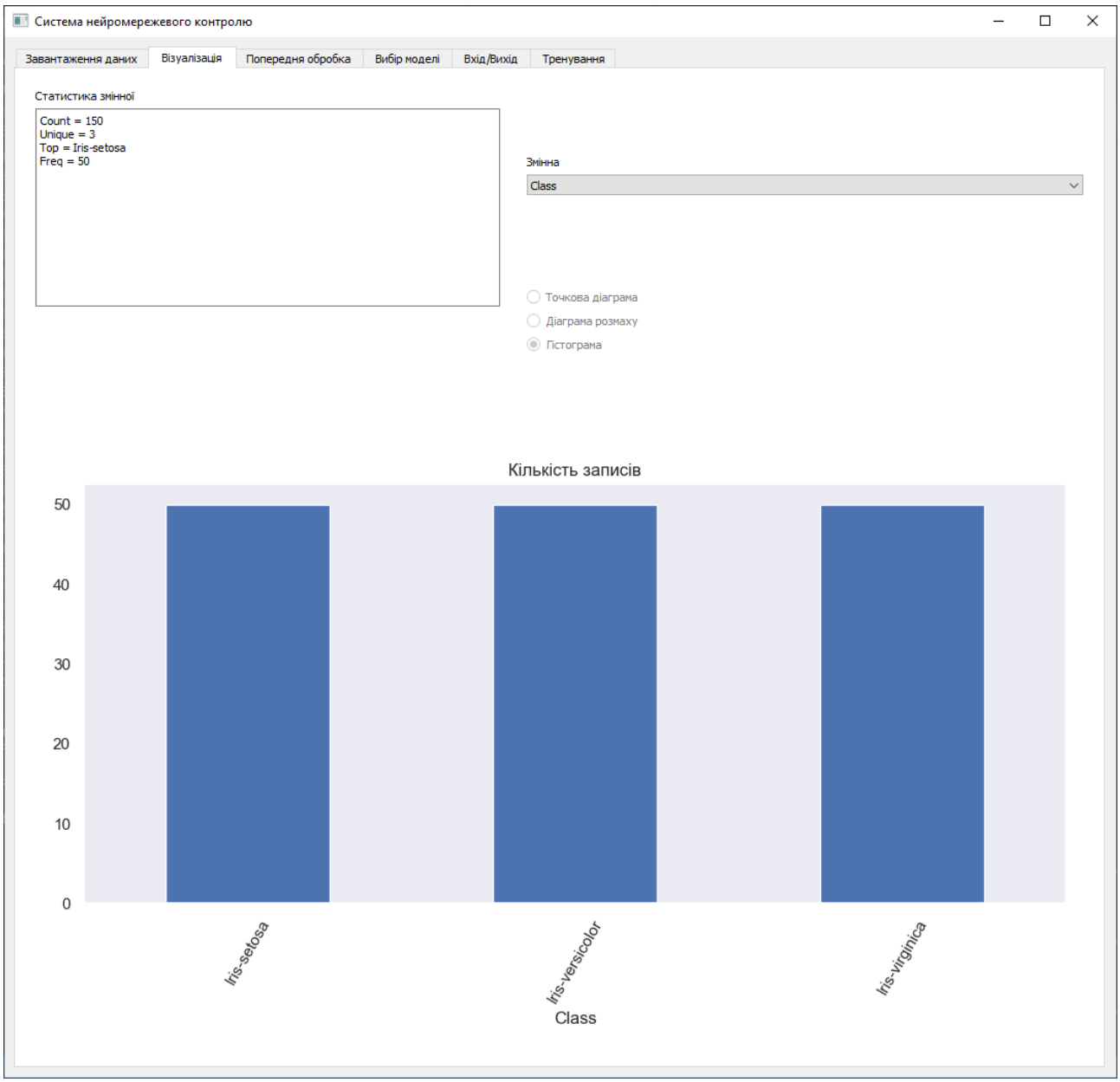
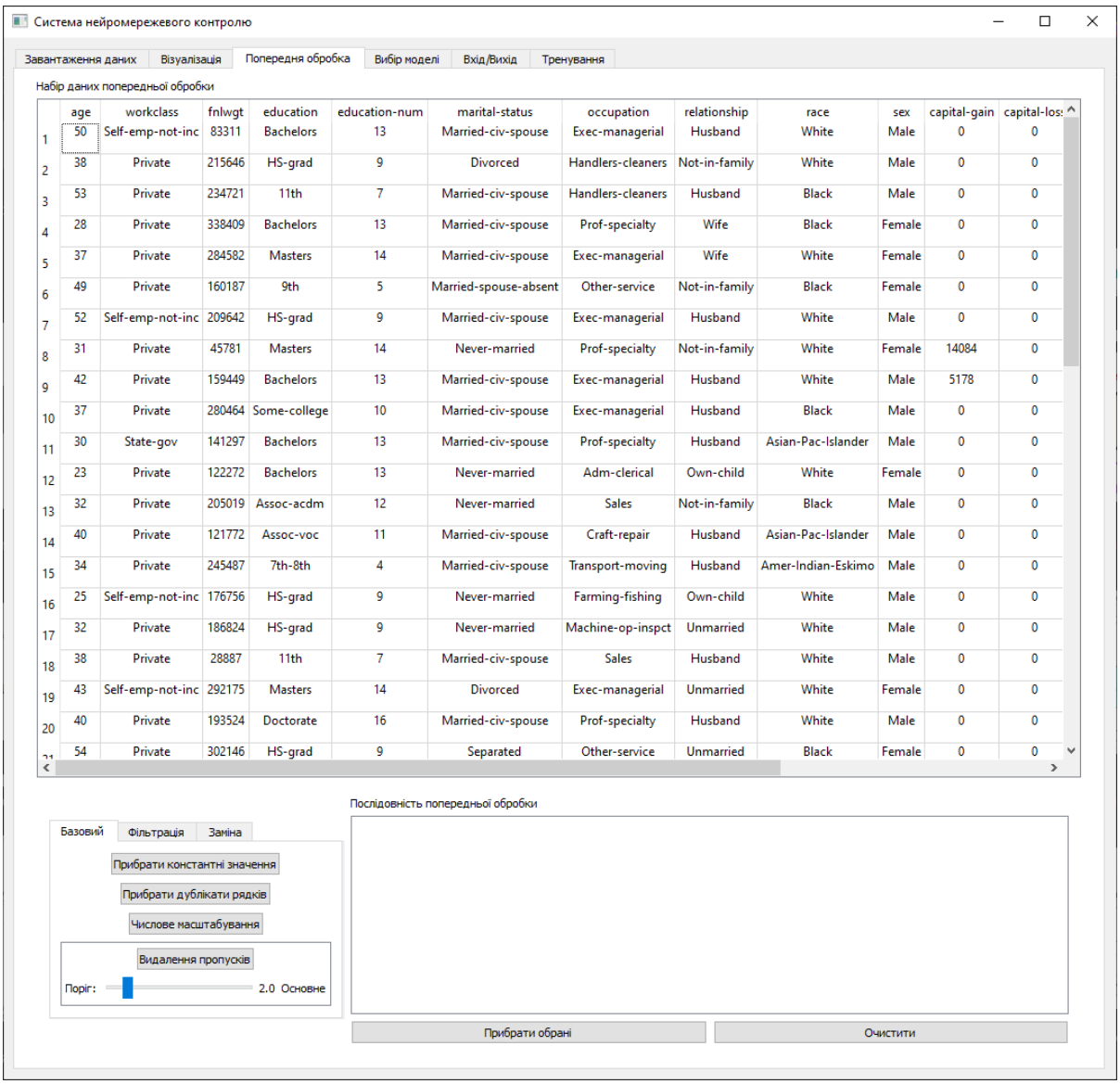


Рисунок 3.5 – Гістограма розподілу класів ірису

3.8. Побудова панелі попередньої обробки даних

Панель обробки дозволяє корегувати набір даних перед тренуванням моделі (рис. 3.6). Попередня обробка даних є ключовим етапом у тренуванні моделей машинного навчання, оскільки забезпечує якість, узгодженість і релевантність вхідної інформації. Вона сприяє зменшенню шуму в даних, усуненню пропущених значень, нормалізації та стандартизації ознак для забезпечення стабільності алгоритмів. Без якісної обробки даних модель може бути чутливою до нерівномірностей у вибірці, втратити здатність узагальнювати результати та демонструвати низьку точність або передбачуваність.



	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-los
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0
5	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0
6	49	Private	160187	9th	5	Married-spouse-absent	Other-service	Not-in-family	Black	Female	0	0
7	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0
8	31	Private	45781	Masters	14	Never-married	Prof-specialty	Not-in-family	White	Female	14084	0
9	42	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	5178	0
10	37	Private	280464	Some-college	10	Married-civ-spouse	Exec-managerial	Husband	Black	Male	0	0
11	30	State-gov	141297	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	Asian-Pac-Islander	Male	0	0
12	23	Private	122272	Bachelors	13	Never-married	Adm-clerical	Own-child	White	Female	0	0
13	32	Private	205019	Assoc-acdm	12	Never-married	Sales	Not-in-family	Black	Male	0	0
14	40	Private	121772	Assoc-voc	11	Married-civ-spouse	Craft-repair	Husband	Asian-Pac-Islander	Male	0	0
15	34	Private	245487	7th-8th	4	Married-civ-spouse	Transport-moving	Husband	Amer-Indian-Eskimo	Male	0	0
16	25	Self-emp-not-inc	176756	HS-grad	9	Never-married	Farming-fishing	Own-child	White	Male	0	0
17	32	Private	186824	HS-grad	9	Never-married	Machine-op-inspct	Unmarried	White	Male	0	0
18	38	Private	28887	11th	7	Married-civ-spouse	Sales	Husband	White	Male	0	0
19	43	Self-emp-not-inc	292175	Masters	14	Divorced	Exec-managerial	Unmarried	White	Female	0	0
20	40	Private	193524	Doctorate	16	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0
21	54	Private	302146	HS-grad	9	Separated	Other-service	Unmarried	Black	Female	0	0

Рисунок 3.6 – Панель попередньої обробки

Панель попередньої обробки має декілька підвкладок. Вони дозволяють прибрати константні значення, що ніколи не змінюються між записами, прибрати ідентичні рядки-дублікати, а також провести числове масштабування.

Масштабування гарантує, що всі ознаки знаходяться в однаковому діапазоні, запобігаючи домінуванню ознак із великими значеннями над іншими. Це сприяє швидшій та стабільнішій оптимізації, особливо для моделей, таких як градієнтний спуск, SVM або k-середніх.

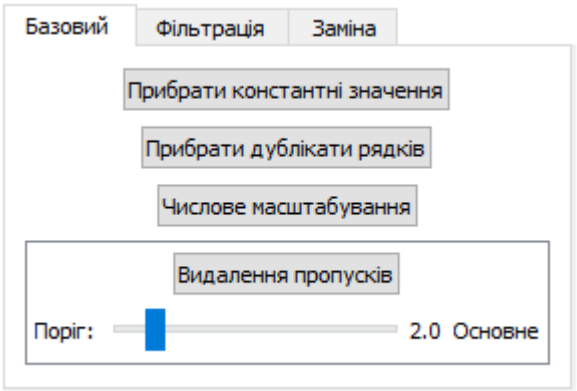


Рисунок 3.7 – Меню базової обробки

Вкладка фільтрації дозволяє обмежити тренування моделі до записів де певна змінна виконує задану користувачем умову. Фільтрація даних може бути корисна для оптимізації моделі під конкретні випадки застосування.

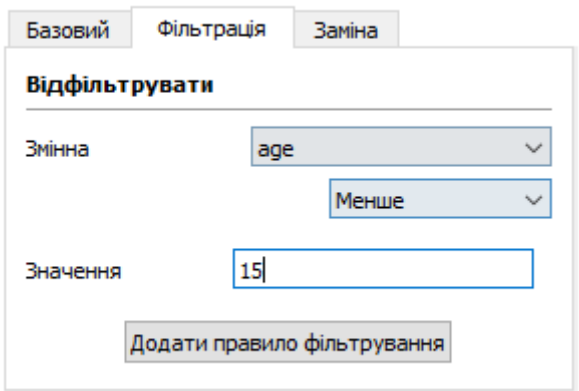


Рисунок 3.8 – Меню фільтрації

Вкладка заміни дозволяє корегувати дані шляхом перезапису полів у вказаних стовпцях. У разі заміни значень словесних змінних, вибір змінної буде обмежено існуючими в наборі для запобігання невдалих операцій.

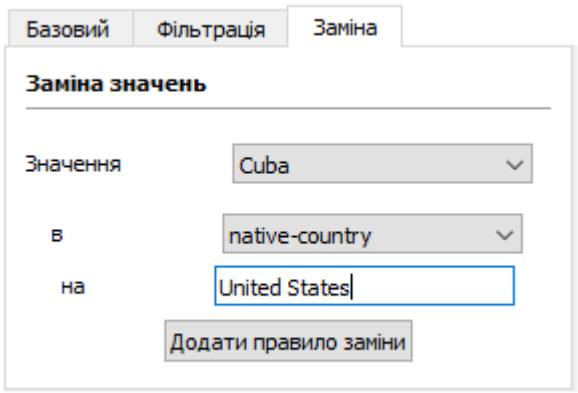


Рисунок 3.9 – Меню заміни

3.9. Побудова панелі вибору параметрів моделі

Панель вибору параметрів дає визначити тип моделі – класифікатор або модель регресії, обрати алгоритм роботи та його параметри.

Першим ключовим вибором є вибір між регресією та класифікацією.

- регресія використовується для прогнозування безперервних числових значень (наприклад, ціна, температура). Вихід моделі є числом;
- класифікація спрямована на передбачення категорій (наприклад, «позитивний/негативний», «клас А/клас В»). Вихід моделі є дискретним (клас).

В залежності від обраного типу моделі, панель вибору вхідних/вихідних даних та панель тренування матимуть різні елементи.

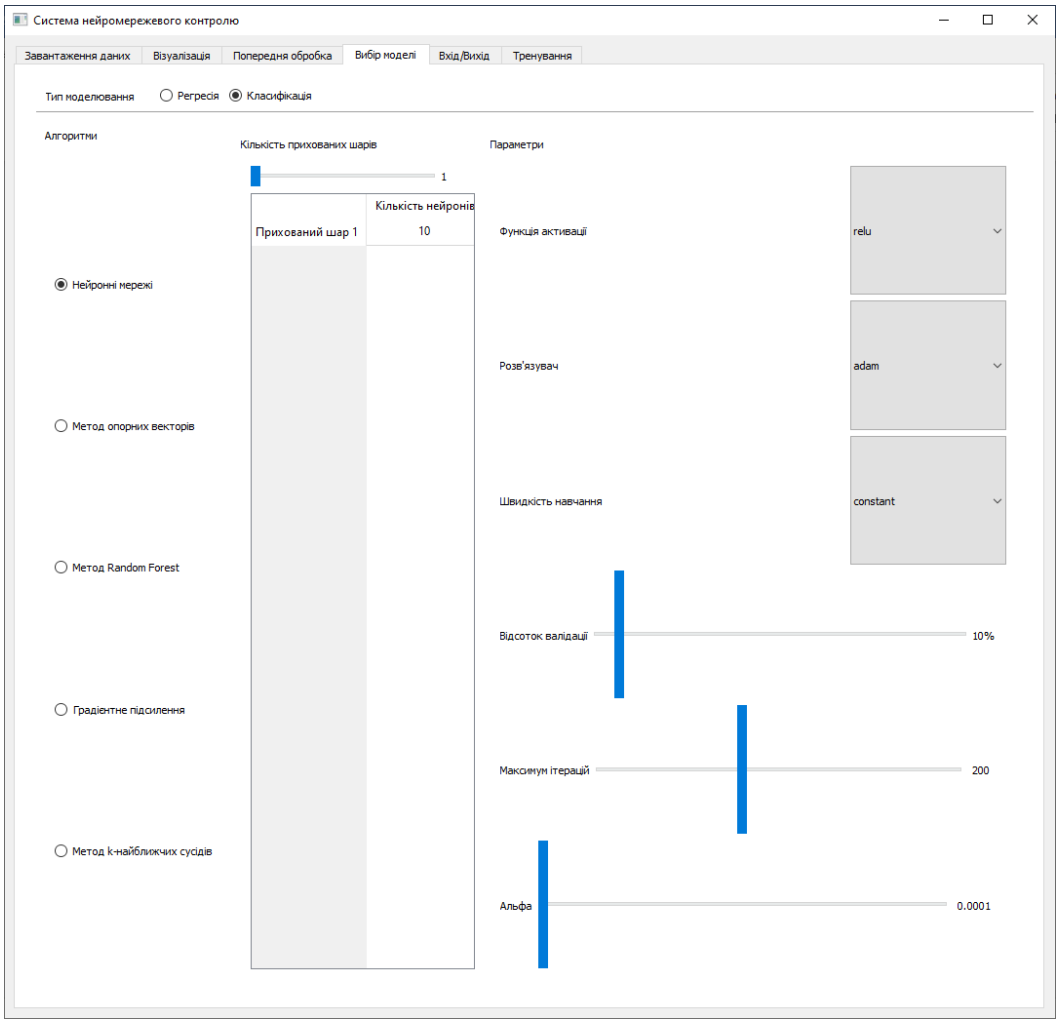


Рисунок 3.10 – Панель вибору моделі

Далі, користувач має обрати алгоритм моделі. Кожен алгоритм має свої переваги і недоліки, тому вибір залежить від характеру задачі, обсягу даних і вимог до точності.

Неправильний вибір моделі може суттєво знизити ефективність системи. Наприклад, використання нейронних мереж для задач із малими обсягами даних часто призводить до перенавчання, тоді як застосування простіших методів, як-от лінійна регресія чи SVM, у складних задачах із нелінійними залежностями може виявитися недостатньо гнучким. Крім того, неадаптовані моделі, як-от k-NN без належного масштабування даних, можуть генерувати некоректні результати через чутливість до діапазону ознак.

Неправильний вибір також впливає на обчислювальну ефективність і

ресурсомісткість. Складні моделі, наприклад градієнтне підсилення чи нейронні мережі, можуть бути надмірними для простих задач, що призведе до втрати часу та ресурсів. Інтерпретованість також страждає: у критичних галузях, таких як медицина, важливо використовувати моделі, які забезпечують зрозумілі прогнози (наприклад, Random Forest). Ретельний аналіз задачі та експерименти з різними моделями допоможуть уникнути таких помилок. [17]

1. Нейронні мережі
   - класифікація: Потужні для складних задач із нелінійними залежностями. Можуть вимагати великої кількості даних і обчислювальних ресурсів.
   - регресія: Ефективні для прогнозування нелінійних залежностей, але можуть перенавчатися на малих обсягах даних.
2. Метод опорних векторів (SVM)
   - класифікація: Добре працює на малих наборах даних із чіткими межами класів. Складно масштабувати для великих наборів.
   - регресія: Підходить для задач, де важлива точність, але менш ефективний для великих або високовимірних даних.
3. Random Forest
   - класифікація: Сильний у задачах із численними категоріями та пропущеними даними. Інтерпретований і стійкий до перенавчання;
   - регресія: Надійний для моделювання середніх значень, але не завжди точний для екстремальних значень.
4. Градієнтне підсилення
   - класифікація: Потужний для багатовимірних даних. Чудово працює в задачах із високою складністю та нерівномірними класами;
   - регресія: Ефективний для точного моделювання складних залежностей і прогнозування, але може бути чутливим до перенавчання.
5. Метод k-найближчих сусідів
   - класифікація: Простий і ефективний для невеликих наборів

даних, але стає повільним на великих наборах;

- регресія: Використовується для прогнозування середнього значення сусідів, проте чутливий до масштабу ознак. [21]

Рисунок 3.11 – Вибір функції активації

Також, для тренування моделі потрібно обрати функцію активації нейронів.

Функція активації в нейронних мережах потрібна для додавання нелінійності в модель, що дозволяє їй вирішувати складні задачі. Вона регулює вихід нейронів, допомагаючи навчати мережу, прискорює процес навчання, а також налаштовує виходи для конкретних задач.

1. ReLU (Rectified Linear Unit):

- класифікація: Добре працює в прихованих шарах завдяки ефективності обчислень і здатності моделювати нелінійності;
- регресія: Підходить для прихованих шарів, але на вихідному шарі не використовується через відсутність обмеження діапазону значень.

2. Identity:

- класифікація: Не підходить для вихідного шару, оскільки не обмежує значення в діапазоні, що ускладнює передбачення ймовірностей;

- регресія: Застосовується у вихідному шарі для прогнозування безперервних числових значень.

3. Logistic (сигмоїда):

- класифікація: Використовується у вихідному шарі для задач бінарної класифікації, забезпечуючи вихідні значення в межах [0, 1];
- регресія: Не використовується через обмежений діапазон значень.

4. Tanh (гіперболічний тангенс):

- класифікація: Використовується в прихованих шарах для моделювання нелінійностей, але не підходить для вихідного шару через значення в [-1, 1];
- регресія: Може застосовуватися в прихованих шарах, але зазвичай не використовується у вихідному шарі через обмеження діапазону.

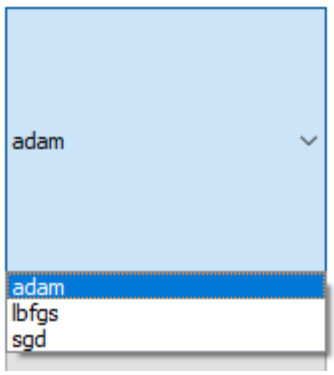


Рисунок 3.12 – Вибір розв’язувача

Для подальшого налаштування моделі, потрібно обрати один з трьох розв’язувачів:

- ADAM (Adaptive Moment Estimation): Адаптивний метод оптимізації, який комбінує переваги двох інших методів (Momentum і RMSprop). Він використовує моменти першого та другого порядку (середні значення градієнтів і їх квадратів) для адаптивної зміни швидкості навчання, що

забезпечує стабільність і швидкість навчання;

- LBFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno): Класичний метод оптимізації, який є варіантом методу Ньютона. Використовує апроксимацію матриці Гессе для ефективного знаходження мінімуму функції з обмеженими ресурсами пам’яті. Підходить для задач з високою точністю, але може бути повільним для великих датасетів;
- SGD (Stochastic Gradient Descent): Стохастичний градієнтний спуск, який оновлює параметри моделі після кожного випадкового підвибору з даних (мішені). Це дозволяє швидше сходитися до мінімуму, але метод може бути шумним і потребує налаштування швидкості навчання.

Рисунок 3.13 – Вибір функції швидкості навчання

Для методів, які використовують градієнтний спуск, таких як логістична регресія чи методи з підтримкою векторів, можна налаштувати тип стратегії для швидкості навчання. Ось короткий опис трьох доступних режимів:

- constant: Швидкість навчання залишається сталою протягом всього процесу навчання (за замовчуванням). Це означає, що на кожному кроці оновлення параметрів застосовується одна і та ж швидкість навчання.
- invscaling: Швидкість навчання зменшується з кожною ітерацією за інверсною пропорцією до числа ітерацій, тобто вона поступово зменшується, що допомагає уникнути переприскорення в кінці навчання.
- adaptive: Швидкість навчання змінюється в залежності від того, як добре модель навчається. Якщо на поточному кроці градієнт не зменшується, швидкість навчання зменшується. Якщо модель добре навчається, швидкість залишається сталою або зростає.

3.10. Побудова панелі вибору вхідних даних

Панель входу/виходу дозволяє обрати змінні що будуть застосовані у тренуванні та змінні, що будуть цільовими. У разі використання моделі класифікатора, цільова змінна обирається з випадаючого списку (рис. 3.14).

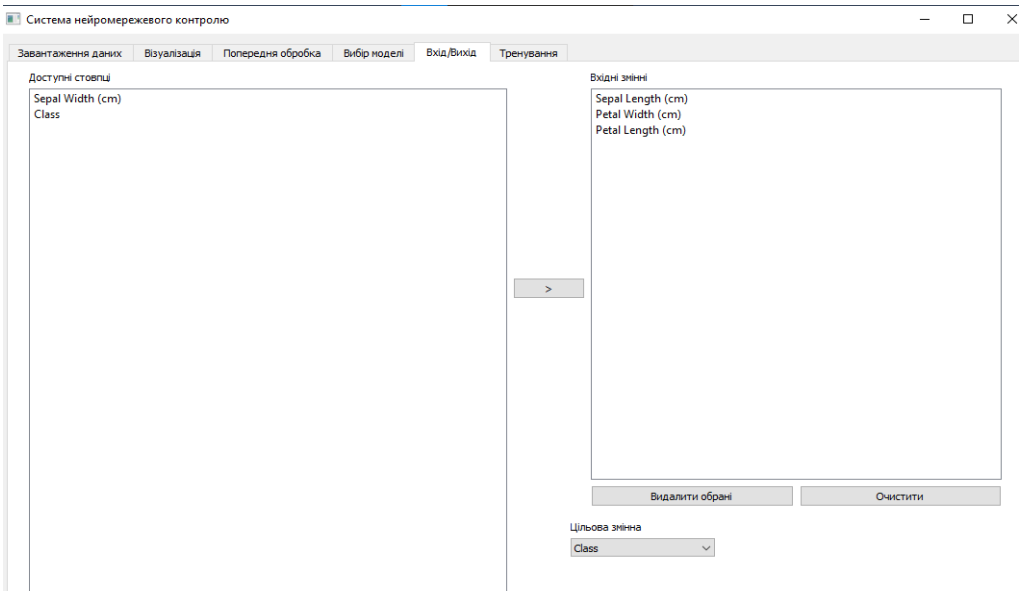


Рисунок 3.14 – Панель вибору змінних класифікації

У разі побудови моделі регресії, можна обрати декілька цільових змінних, перемістивши їх у відповідне поле (рис. 3.15).

Змінні, що залишились у полі доступних стовпців будуть упущені та не застосовуватимуться для тренування моделі.

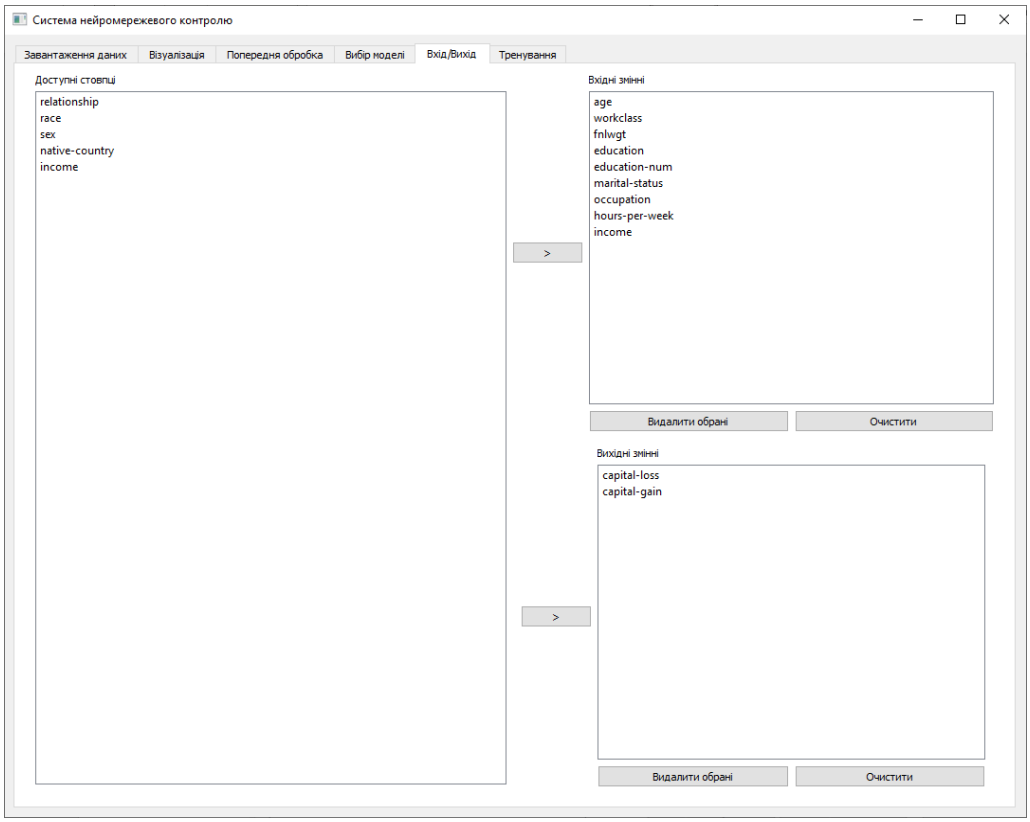


Рисунок 3.15 – Панель вибору змінних регресії

3.11. Побудова панелі тренування моделі

Панель тренування дозволяє почати процес тренування, виводу статистики та збереження моделі. Перед початком тренування, слід визначити пропорцію розподілу набору даних на тестувальний та тренувальний набори. Це є необхідною умовою для забезпечення коректної оцінки якості моделі та запобігання перенавчанню. Використання однакових даних для тренування та тестування призводить до того, що модель може просто запам’ятовувати специфічні особливості даного набору, що унеможливлює її здатність до узагальнення. Це, в свою чергу, негативно впливає на її здатність ефективно прогнозувати на нових, невідомих даних.

Тестувальний набір є основою для об’єктивної оцінки здатності моделі до узагальнення, а також для вибору оптимальної архітектури та параметрів моделі.

Він дозволяє перевірити, наскільки добре модель здатна переносити знання, отримані на тренувальних даних, на нові спостереження, що не використовувалися в процесі навчання, і таким чином є ключовим інструментом для забезпечення високої прогностичної здатності моделі.

Також, перед тренуванням слід визначити, чи необхідно перемішати зразки записів із набору даних. Перемішування зразків перед тренуванням нейронної мережі є важливим для запобігання впливу порядку подачі даних на навчання моделі. Без перемішування модель може навчатися на зразках у певній послідовності, що може призвести до систематичних помилок, особливо якщо дані містять якісь структурні залежності чи закономірності.

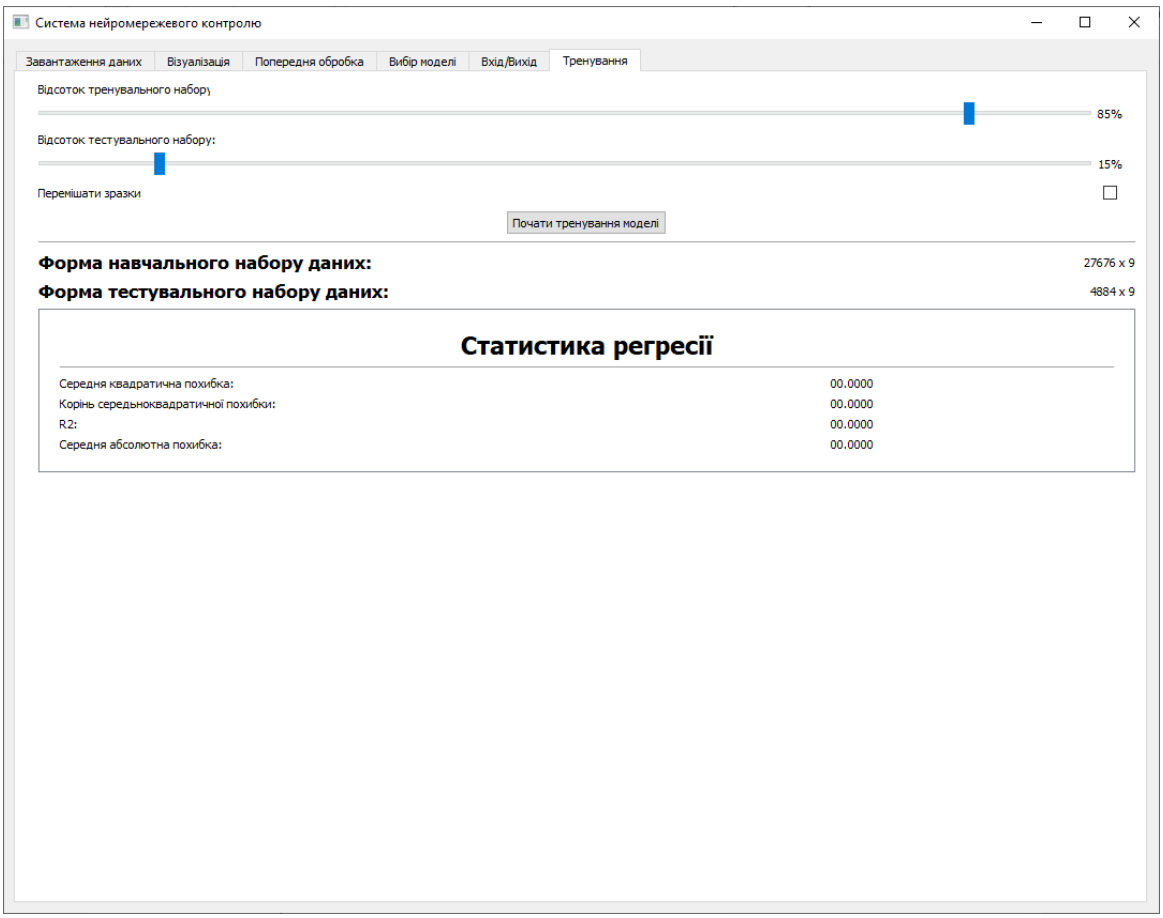


Рисунок 3.16 – Панель тренування

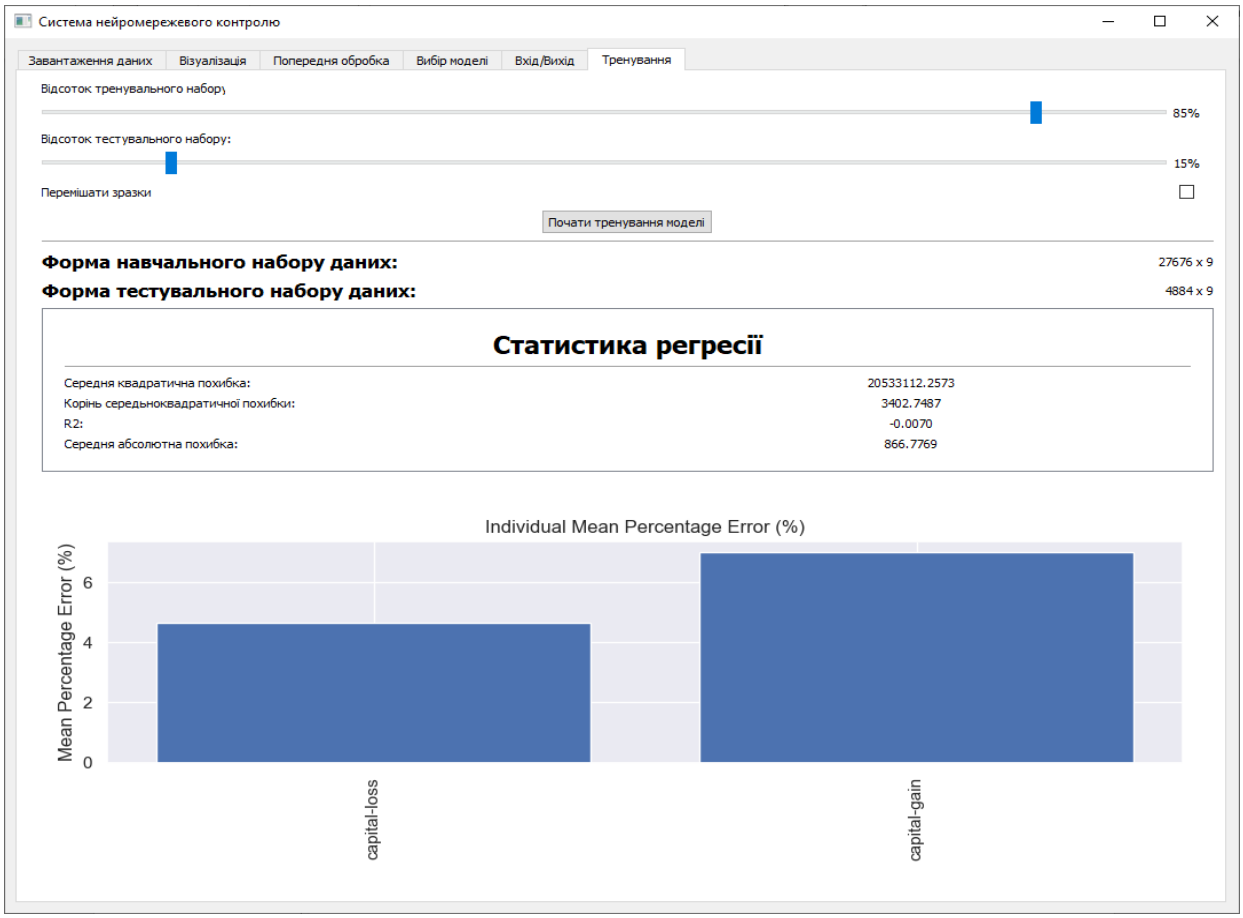


Рисунок 3.17 – Результати тренування регресії

По завершенню тренування на екран виводяться показники ефективності моделі: середня квадратична похибка, корінь середньоквадратичної похибки, R^2, середня абсолютна похибка Також, візуалізується індивідуальна похибка для кожної змінної.

Individual Mean Percentage Error (IMPE) є статистичним показником для оцінювання точності регресійної моделі. Він вимірює середнє відносне відхилення прогнозованих значень від фактичних, виражене у відсотках. IMPE дозволяє оцінити, наскільки ефективно модель адаптується до змін у даних та наскільки точні її прогнози відносно реальних значень, незалежно від масштабу або одиниць вимірювання змінної. Метрика чутлива до нульових або близьких до нуля значень цільової змінної, що може призводити до некоректних інтерпретацій.

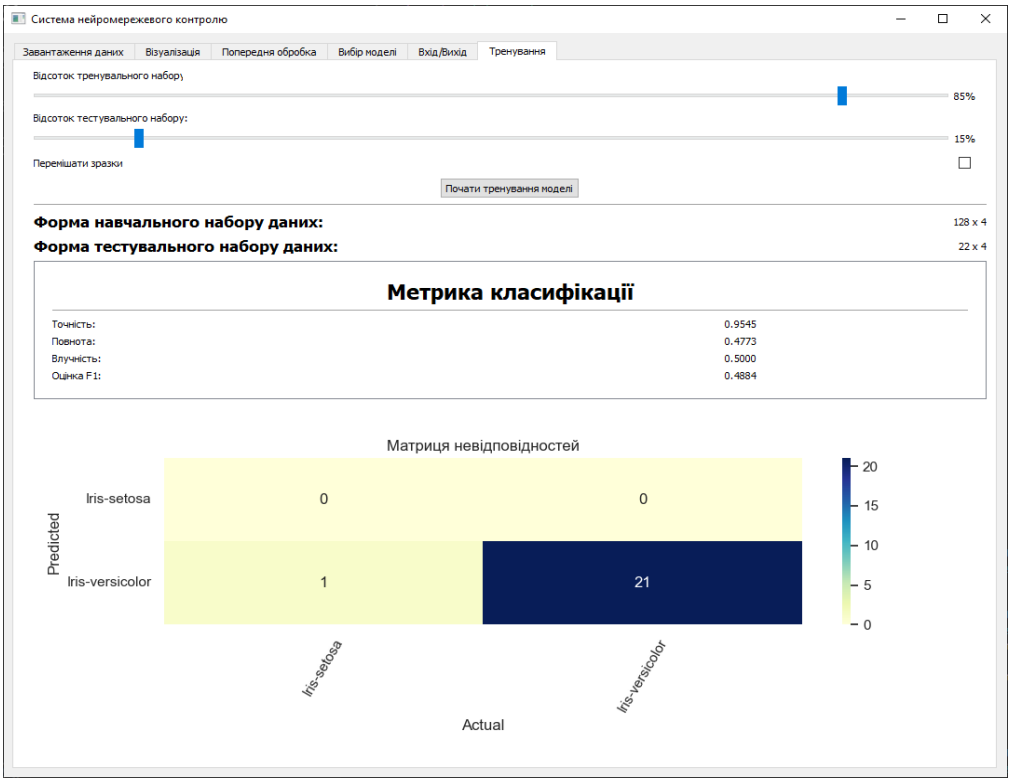


Рисунок 3.18 – Результати тренування моделі класифікації

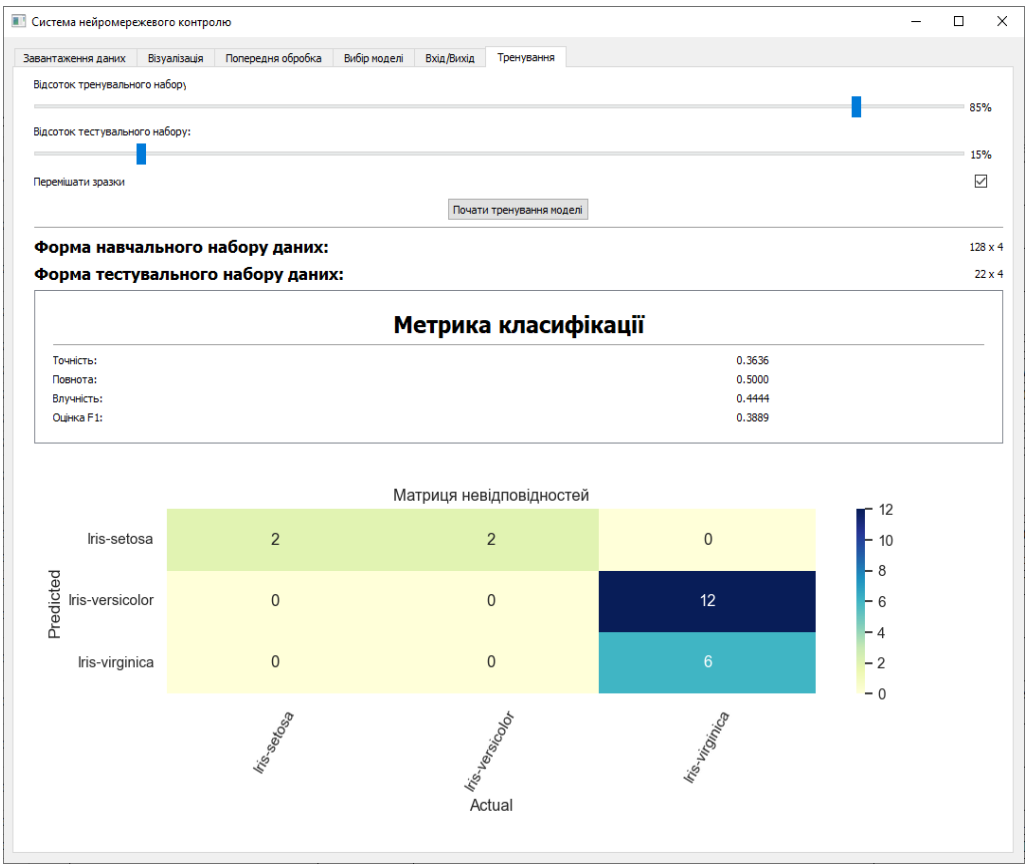


Рисунок 3.19 – Результати тренування моделі зі змішуванням змінних

По завершенню тренування, створена модель зберігається у окремому файлі, дозволяючи використовувати її для подальшої роботи з даними у інших застосунках.

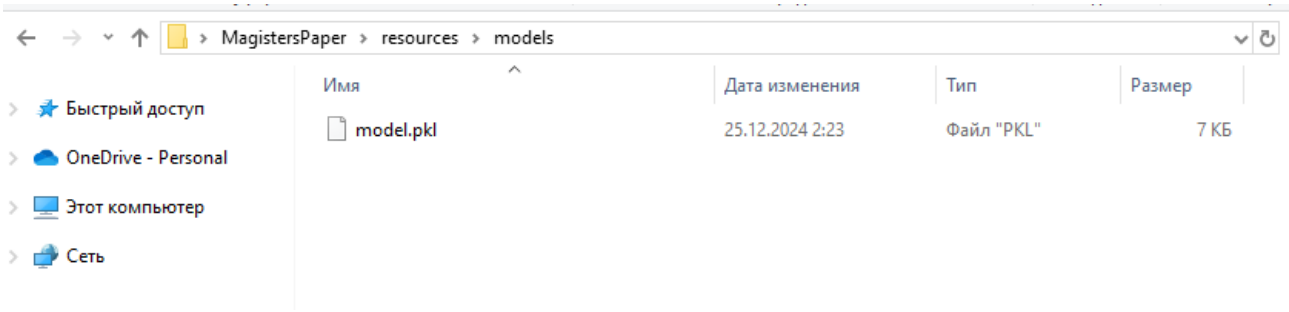


Рисунок 3.20 – Збережена модель

РОЗДІЛ 4
ТЕСТУВАННЯ ЗАСТОСУНКУ

Тестування застосунку проводиться для встановлення можливих помилок у роботі та забезпечення стабільного користування програмою. Тестування додатку виконується за створеними тест-кейсами (табл. 4.1-4.2).

Тест-кейс (англ. Test case) – це набір дій, які виконуються над системою, щоб визначити, чи задовольняє вона програмним вимогам і чи функціонує правильно.

Мета тестового кейсу – визначити, чи працюють різні функції в системі так, як очікується, і підтвердити, що система задовольняє всім відповідним стандартам, інструкціям і вимогам замовника. Процес написання тестового кейсу також може допомогти виявити помилки або дефекти в системі.

Тестові кейси повинні бути розроблені таким чином, щоб повністю відображати особливості та функціональність оцінюваного програмного додатку. Структура кейсу повинна бути зроблена так, щоб за один раз тестувалася тільки одна річ. Мова, що використовується для написання тестових кейсів, повинна бути простою і зрозумілою, активною, а не пасивною, а також точною і послідовною при іменуванні елементів.

Компоненти тестового кейсу включають:

- назва тесту. Назва, яка описує функціональність або властивість, яку перевіряє тест;
- ідентифікатор тесту. Зазвичай це цифровий або буквено-цифровий ідентифікатор, який використовують для групування тестових кейсів у набори тестів;
- мета. Описує те, що має перевірити тест, в одному-двох реченнях;
- посилання. Посилання на історії користувачів, специфікації дизайну або вимоги, які потрібно перевірити;
- передумови. Будь-які умови, необхідні для виконання тесту у застосунку;

- середовище тесту. Цей компонент визначає, що потрібно для коректного запуску тестового кейсу, наприклад, версія програми, операційна система, вимоги до дати і часу, а також специфікації безпеки;
- етапи тестування. Детальний опис послідовних дій, які необхідно виконати для завершення тесту;
- очікувані результати. Опис того, як система повинна реагувати на кожен крок тесту. [22]

Таблиця 4.1 – Тест-кейс завантаження файлу

Опис	Перевірка завантаження файлів некоректного типу при спробі обрати набір даних	
Середовище	Персональний комп’ютер з ОС Windows 10, Windows 11	
Передумови	Відкрито головна панель програми	
№	Дія	Очікуваний результат
1	Натиснути кнопку «Імпорт файлу»	Відображено вікно провідника файлової системи.
2	Обрати файл	Фон файл підсвічується іншим кольором.
3	Підтвердити вибір	Вміст файлу відображається в програмі або виводиться повідомлення про помилку імпорту.

Під час виконання алгоритму тест кейсу, було отримано попередження про помилку вводу (рис 4.1).

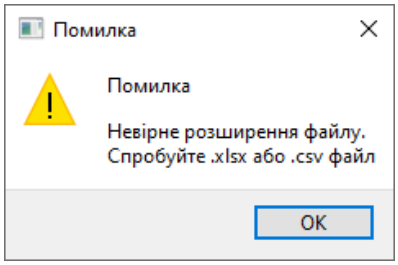


Рисунок 4.1 – Повідомлення про помилку вводу

Таблиця 4.2 – Тест-кейс тренування моделі

Опис	Перевірка тренування моделі з неправильно обраними змінними	
Середовище	Персональний комп’ютер з ОС Windows 10, Windows 11	
Передумови	Відкрито панель тренування програми	
№	Дія	Очікуваний результат
1	Відкрити панель вибору вводу	Відображено список змінних.
2	Очистити список вихідних змінних	Кнопка тренування стає недоступною для натискання.
3	Натиснути кнопку тренування	Тренування не буде розпочато.

Таблиця 4.3 – Тест-кейс візуалізації даних

Опис	Перевірка візуалізації даних з неправильно обраним типом графіку	
Середовище	Персональний комп’ютер з ОС Windows 10, Windows 11	
Передумови	Відкрито панель візуалізації програми	
№	Дія	Очікуваний результат
1	Завантажити набір даних з словесними змінними	Змінна додана до списку на панелі візуалізації
2	Обрати лінійну діаграму	Вибір діаграми візуально відображено
3	Обрати словесну змінну для відображення	Діаграма замінюється на відповідний змінній тип

ВИСНОВОК

Була побудована згорткова модель нейронних мереж для визначення класів ураження шкіри за допомогою мови програмування Python на основі бібліотеки Scikit-Learn та ПЗ QT Designer. Було створено десктопний застосунок з графічним інтерфейсом для ОС Windows, що може дозволяє керувати процесом тренування нейронних моделей та отримувати статистику про їх ефективність. Було оцінено можливі помилки при тренуванні моделі та створено інтерфейс для виведення метрики імовірних збитків.

В якості вхідних даних моделі були використані бази даних відкритого репозиторію University of California Irvine (UCI).

У якості вихідних даних програма зберігає створену модель окремим файлом та відображає статистику моделі. Були розглянуті шляхи оптимізації процесів навчання та експлуатації моделі. Було розглянуто недоліки та переваги використання різних параметрів при тренуванні моделі.

Було встановлено, що програма здатна успішно створювати та готувати до експорту моделі класифікатори та моделі регресії. Для подальшого підвищення ефективності роботи запропоновано розширити набір доступних алгоритмів тренування та створити систему автоматичного вибору параметрів тренування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DeVore, Ronald, Boris Hanin, and Guergana Petrova. “Neural Network Approximation.” Acta Numerica 30 (2021): 327–444. doi:10.1017/S0962492921000052. URL: https://arxiv.org/abs/2012.14501 (дата звернення: 06.12.2024).

2. Application of Generative Diffusion Models in digital image creation / G. Golovko, O. Bilokin. // Системи управління, навігації та зв’язку. Збірник наукових праць. – Полтава: ПНТУ, 2022. – Т. 4 (70). – С. 71-74.

3. Sarker, I.H. Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN COMPUT. SCI.* **2**, 160 (2021). URL: https://doi.org/10.1007/s42979-021-00592-x (дата звернення: 06.12.2024).

4. J. Straub, Expert system gradient descent style training: Development of a defensible artificial intelligence technique, Knowledge-Based Systems (2021), doi: https://doi.org/10.1016/j.knosys.2021.107275 . (дата звернення: 06.12.2024).

5. Fatima, M., & Pasha, M. (2017). Survey of machine learning algorithms for disease diagnostic. Journal of Intelligent Learning Systems and Applications, 9(01) – С. 1-16.

6. Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of artificial intelligence research, *4* – С. 237-285.

7. Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: a system for large-scale machine learning. In Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation (OSDI’16). USENIX Association, USA, 265–283. URL: https://arxiv.org/abs/1603.04467 (дата звернення: 07.12.2024)

8. Model optimization | TensorFlow Lite URL: https://www.tensorflow.org/lite/performance/model_optimization (дата звернення: 07.12.2024)

9. GitHub - keras-team/keras: Deep Learning for humans URL:

https://github.com/keras-team/keras (дата звернення: 07.12.2024)

10. ISIC | International Skin Imaging Collaboration URL: https://challenge.isic-archive.com/ (дата звернення: 07.12.2024)

11. Mota AN, Piñeiro-Maceira J, Alves Mde F, Tarazona MJ. Pigmented Bowen's disease. An Bras Dermatol. 2014 Sep-Oct;89(5):825-7. doi: 10.1590/abd1806-4841.20142725. PMID: 25184929; PMCID: PMC4155968.

12. About Anaconda | Anaconda humans URL: https://www.anaconda.com/about-us (дата звернення: 07.12.2024)

13. CUDA GPUs - Compute Capability | NVIDIA Developer URL: https://developer.nvidia.com/cuda-gpus (дата звернення: 07.12.2024)

14. Augmentor — Augmentor 0.2.12 documentation URL: https://augmentor.readthedocs.io/en/master/ (дата звернення: 07.12.2024)

15. Download Android Studio & App Tools - Android Developers URL: https://developer.android.com/studio (дата звернення: 07.12.2024)

16. Project Jupyter Documentation — Jupyter Documentation 4.1.1 alpha documentation URL: https://docs.jupyter.org/en/latest/ (дата звернення: 07.12.2024)

17. Paris, Grégory, Denis Robilliard and Cyril Fonlupt. "Exploring Overfitting in Genetic Programming." Artificial Evolution (2003). URL: https://api.semanticscholar.org/CorpusID:39887373 (дата звернення: 07.12.2024)

Vision Solutions. Assessing the Financial Impact of Downtime. White paper, 2014. URL: https://luminet.co.uk/wp-content/uploads/2014/10/Assessing-the-Financial-Impact-of-Downtime-UK.pdf. (дата звернення: 07.12.2024)

19. BM Subraya and SV Subrahmanya. Object driven performance testing of web applications. In Quality Software, 2000. Proceedings. First Asia-Pacific Conference on, pages 17–26. IEEE, 2000.

20. E.J. Weyuker and F.I. Vokolos. Experience with performance testing of software systems: issues, an approach, and case study. Software Engineering, IEEE Transactions on, 26(12):1147–1156, 2000

21. Zheng H, Zhou Z, Chen J. RLSTM: A New Framework of Stock

Prediction by Using Random Noise for Overfitting Prevention. Comput Intell Neurosci. 2021 May 19;2021:8865816. doi: 10.1155/2021/8865816. PMID: 34113377; PMCID: PMC8154285. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8154285/ (дата звернення: 08.12.2024)

22. Ying, X. (2019). An Overview of Overfitting and its Solutions. Journal of Physics: Conference Series, 1168, 022022. doi:10.1088/1742-6596/1168/2/022022

23. Хайкін С. Нейронні мережі: повний курс, 2-е видання. Пер. з англ. - М.: Видавничий дім «Вільямс», 2006 р. - 1104с..

24. Байєсівські мережі в системах підтримки прийняття рішень [Електронний ресурс] : навчальний посібник / М. З. Згуровський, П. І. Бідюк, О. М. Терентьєв, Т. І. Просянкіна-Жарова ; НТУУ «КПІ». Київ : ТОВ «Видавниче Підприємство «Едельвейс», 2015. – 300 с. URL: https://ela.kpi.ua/handle/123456789/19582 (дата звернення: 08.12.2024)

25. Långström, S. (2021). An Investigative Study of Testing Strategy and Test Case Creation in a Hardware-Software Co-design Environment Using Software Product Line Theory URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-305219 (дата звернення: 08.12.2024)

26. ML.NET | Machine learning made for .NET URL: https://dotnet.microsoft.com/en-us/apps/machinelearning-ai/ml-dotnet (дата звернення: 08.12.2024)

27. UCI Machine Learning Repository URL: http://archive.ics.uci.edu/

28. A. Unwin, K. Kleinman (2021) The iris data set: In search of the source of virginica URL: https://www.semanticscholar.org/paper/The-iris-data-set%3A-In-search-of-the-source-of-Unwin-Kleinman/4599862ea877863669a6a8e63a3c707a787d5d7e (дата звернення: 07.12.2024)

29. Why Visual Studio Code? – Overview URL: https://code.visualstudio.com/docs/editor/whyvscode (дата звернення: 07.12.2024)

30. pyuic5-tool · PyPI description URL: https://pypi.org/project/pyuic5-tool/ (дата звернення: 12.12.2024)

31. Документація по .NET | Microsoft Learn URL: https://learn.microsoft.com/ru-ru/dotnet/core/introduction (дата звернення: 12.12.2024)

ДОДАТОК А
ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ

Файл controller.py

```
import os
import random
import sys
import threads
import seaborn as sns
from os.path import join, abspath
from view import QtCore, QtWidgets
import personalised_widgets

# Назначення параметрів seaborn
sns.set()


class ViewController:

    def __init__(source, ui, ml_model):
        source.ui = ui
        source.ml_model = ml_model

        source.root_directory = get_project_root_directory()
        source.src_directory = get_project_root_directory()+'src/'
        source.data_directory = source.root_directory+'data/'

        source.configure_gui()

    def configure_gui(source):

        ui = source.ui
        _translate = QtCore.QCoreApplication.translate

        # Підготовка нового потоку
        ui.threadpool = QtCore.QThreadPool()
        # Наповнення example_dataset_comboBox
        ui.example_dataset_comboBox.addItem('', '')
        list_of_datasets =
os.listdir(transform_to_resource_path(source.data_directory))
        for dataset in list_of_datasets:
            if not dataset.startswith('.'):
                #  Кожен об'єкт отримує назву набору даних в якості тексту і шлях
як дані
                ui.example_dataset_comboBox.addItem(dataset.split('.')[0],
source.data_directory + dataset)
```

```
        source.connect_signals()

        ui.nn_classification_radioButton.click()
        ui.nn_regression_radioButton.click()
        ui.nn_regression_radioButton.click()

        ui.tabs_widget.setCurrentIndex(0)
        ui.pre_process_tabWidget.setCurrentIndex(0)
        ui.output_selection_stackedWidget.setCurrentIndex(0)

        # Вимкнення віджетів якщо немає даних
        widgets_to_disable = [ui.plot_radioButton, ui.boxplot_radioButton,
ui.histogram_radioButton,
                              ui.remove_duplicates_pushButton,
ui.remove_constant_variables_pushButton,
                              ui.numeric_scaling_pushButton,
ui.remove_outliers_pushButton,
                              ui.addrule_filter_value_pushButton,
ui.addrule_replace_value_pushButton,
                              ui.addrule_filter_value_pushButton,
                              ui.add_input_columns_pushButton,
ui.add_output_columns_pushButton,
                              ui.train_model_pushButton,
ui.remove_preprocessing_rule_pushButton,
                              ui.clear_preprocessing_rule_pushButton]

        for widget in widgets_to_disable:
            widget.setEnabled(False)

        ui.spinner_traning_results =
personalised_widgets.QtWaitingSpinner(ui.model_train_widget)
        ui.spinner_traning_results.setSizePolicy(ui.model_train_widget.sizePolicy()
)

    def connect_signals(source):
        ui = source.ui

        # Під'єднання вкладки завантаження файлу
        ui.load_file_pushButton.clicked.connect(lambda:
source.trigger_loading_dataset_thread(ui.load_file_pushButton))
        ui.example_dataset_comboBox.currentIndexChanged.connect(
            lambda:
source.trigger_loading_dataset_thread(ui.example_dataset_comboBox))

        # Під'єднання вкладки візуалізації
        ui.variable_to_plot_comboBox.currentIndexChanged.connect(lambda:
source.update_visualisation_options())
```

```
        ui.boxplot_radioButton.clicked.connect(lambda:
source.update_visualisation_widgets())
        ui.plot_radioButton.clicked.connect(lambda:
source.update_visualisation_widgets())
        ui.histogram_radioButton.clicked.connect(lambda:
source.update_visualisation_widgets())

        ui.remove_duplicates_pushButton.clicked.connect(lambda:
source.add_rm_duplicate_rows_rule())
        ui.remove_constant_variables_pushButton.clicked.connect(lambda:
source.add_rm_constant_var_rule())
        ui.numeric_scaling_pushButton.clicked.connect(lambda:
source.add_num_scaling_rule())
        ui.remove_outliers_pushButton.clicked.connect(lambda:
source.add_rm_outliers_rule())
        ui.addrule_filter_value_pushButton.clicked.connect(lambda:
source.generate_filtering_rule())
        ui.addrule_replace_value_pushButton.clicked.connect(lambda:
source.generate_replacing_rule())

        neuros_table_regression = ui.reg_nn_layers_tableWidget
        neuros_table_regression.cellChanged.connect(lambda:
source.check_neurons_number(neuros_table_regression))
        neuros_table_classification = ui.clas_nn_layers_tableWidget
        neuros_table_classification.cellChanged.connect(lambda:
source.check_neurons_number(neuros_table_classification))

        ui.outliers_treshold_horizontalSlider.valueChanged.connect(
            lambda:
source.update_label_from_slider_change(ui.outliers_treshold_horizontalSlider.value(
),
                                                   ui.outliers_treshold_label
))

        ui.replace_columnSelection_comboBox.currentIndexChanged.connect(
            lambda: source.update_preprocess_replace_fields())
        ui.filter_columnSelection_comboBox.currentIndexChanged.connect(
            lambda: source.update_preprocess_filtering_fields())

        ui.add_input_columns_pushButton.clicked.connect(
            lambda:
source.update_input_output_columns(ui.input_columns_listWidget))
        ui.add_output_columns_pushButton.clicked.connect(
            lambda:
source.update_input_output_columns(ui.output_columns_listWidget))

        ui.remove_input_columns_pushButton.clicked.connect(
            lambda:
source.remove_item_from_listwidget(ui.input_columns_listWidget))
        ui.remove_output_columns_pushButton.clicked.connect(
```

```
            lambda:
source.remove_item_from_listwidget(ui.output_columns_listWidget))
        ui.remove_preprocessing_rule_pushButton.clicked.connect(
            lambda:
source.remove_item_from_listwidget(ui.preprocess_sequence_listWidget))

        ui.clear_input_columns_pushButton.clicked.connect(lambda:
source.clear_listwidget(ui.input_columns_listWidget))
        ui.clear_output_columns_pushButton.clicked.connect(lambda:
source.clear_listwidget(ui.output_columns_listWidget))
        ui.clear_preprocessing_rule_pushButton.clicked.connect(lambda:
source.clear_listwidget(ui.preprocess_sequence_listWidget))

        model_selection_radio_buttons = [ui.regression_selection_radioButton,
ui.classification_selection_radioButton,
                                         ui.gradientboosting_classification_radioBu
tton,
                                         ui.knn_classification_radioButton,
                                         ui.nn_classification_radioButton,
ui.randomforest_classification_radioButton,
                                         ui.svm_classification_radioButton,
ui.svm_regression_radioButton,
                                         ui.randomforest_regression_radioButton,
ui.nn_regression_radioButton,
                                         ui.gradientboosting_regression_radioButton
]

        for model_option in model_selection_radio_buttons:
            model_option.clicked.connect(lambda:
source.model_selection_tab_events())

        ui.reg_nn_layers_horizontalSlider.valueChanged.connect(
            lambda:
source.update_label_from_slider_change(ui.reg_nn_layers_horizontalSlider.value(),
                                                              ui.reg_nn_layers_label))
        ui.clas_nn_val_percentage_horizontalSlider.valueChanged.connect(
            lambda:
source.update_label_from_slider_change(ui.clas_nn_val_percentage_horizontalSlider.v
alue(),
                                                              ui.reg_nn_val_percent_labe
l))
        ui.reg_nn_max_iter_horizontalSlider.valueChanged.connect(
            lambda:
source.update_label_from_slider_change(ui.reg_nn_max_iter_horizontalSlider.value(),
                                                              ui.reg_nn_max_iter_label))
        ui.reg_nn_alpha_horizontalSlider.valueChanged.connect(
            lambda:
source.update_label_from_slider_change(ui.reg_nn_alpha_horizontalSlider.value(),
                                                              ui.reg_nn_alpha_label))
        ui.clas_nn_layers_horizontalSlider.valueChanged.connect(
```

```
            lambda:
source.update_label_from_slider_change(ui.clas_nn_layers_horizontalSlider.value(),
                                                   ui.clas_nn_layers_label))
        ui.reg_nn_val_percentage_horizontalSlider.valueChanged.connect(
            lambda:
source.update_label_from_slider_change(ui.reg_nn_val_percentage_horizontalSlider.va
lue(),
                                                   ui.clas_nn_val_percent_lab
el))
        ui.clas_nn_max_iter_horizontalSlider.valueChanged.connect(
            lambda:
source.update_label_from_slider_change(ui.clas_nn_max_iter_horizontalSlider.value()
,
                                                   ui.clas_nn_max_iter_label)
)
        ui.clas_nn_alpha_horizontalSlider.valueChanged.connect(
            lambda:
source.update_label_from_slider_change(ui.clas_nn_alpha_horizontalSlider.value(),
                                                   ui.clas_nn_alpha_label))
        ui.train_percentage_horizontalSlider.valueChanged.connect(
            lambda:
source.update_label_from_slider_change(ui.train_percentage_horizontalSlider.value()
,
                                                   ui.train_percentage_label)
)
        ui.test_percentage_horizontalSlider.valueChanged.connect(
            lambda:
source.update_label_from_slider_change(ui.test_percentage_horizontalSlider.value(),
                                                   ui.test_percentage_label))

        ui.clas_svm_kernel_comboBox.currentIndexChanged.connect(
            lambda: source.update_svm_model_parameters('kernel_change', False))
        ui.clas_svm_C_horizontalSlider.valueChanged.connect(
            lambda: source.update_svm_model_parameters('regularisation_change',
False))
        ui.clas_svm_maxiter_nolimit_checkBox.clicked.connect(
            lambda: source.update_svm_model_parameters('no_limit_click', False))
        ui.clas_svm_maxiter_horizontalSlider.valueChanged.connect(
            lambda: source.update_svm_model_parameters('max_iter_change', False))
        ui.reg_svm_kernel_comboBox.currentIndexChanged.connect(
            lambda: source.update_svm_model_parameters('kernel_change', True))
        ui.reg_svm_C_horizontalSlider.valueChanged.connect(
            lambda: source.update_svm_model_parameters('regularisation_change',
True))
        ui.reg_svm_episilon_horizontalSlider.valueChanged.connect(
            lambda: source.update_svm_model_parameters('epsilon_change', True))
        ui.reg_svm_maxiter_nolimit_checkBox.clicked.connect(
            lambda: source.update_svm_model_parameters('no_limit_click', True))
        ui.reg_svm_maxiter_horizontalSlider.valueChanged.connect(
            lambda: source.update_svm_model_parameters('max_iter_change', True))
```

```
        ui.tabs_widget.currentChanged.connect(lambda:
source.update_input_output_columns('clear_output_variables'))

        ui.train_model_pushButton.clicked.connect(lambda:
source.trigger_train_model_thread())

    def trigger_loading_dataset_thread(source, data_source):
        ui = source.ui
        ml_model = source.ml_model

        ui.dataset_tableWidget.spinner.start()
        ui.pre_process_dataset_tableWidget.spinner.start()

        if data_source.objectName() == 'load_file_pushButton':
            fileDlg = QtWidgets.QFileDialog()
            file_address = fileDlg.getOpenFileName()[0]

        elif data_source.objectName() == 'example_dataset_comboBox':
            selected_index = ui.example_dataset_comboBox.currentIndex()
            file_address = ui.example_dataset_comboBox.itemData(selected_index)

            # Видалення порожнього запису
            if ui.example_dataset_comboBox.itemText(0) == '':
                ui.example_dataset_comboBox.blockSignals(True)
                ui.example_dataset_comboBox.removeItem(0)
                ui.example_dataset_comboBox.blockSignals(False)

        if file_address == '' or file_address == None:
            ui.dataset_tableWidget.spinner.stop()
            ui.pre_process_dataset_tableWidget.spinner.stop()
            return

        ui.load_file_pushButton.setDisabled(True)
        ui.example_dataset_comboBox.setDisabled(True)

        # Створення обробщика об'єктів
        worker = threads.Load_Dataset_Thread(ui, ml_model, file_address)

        # Під'єднання сигналів до виклику функцій
        worker.signals.stop_spinner.connect(source.update_table_widget)
        worker.signals.display_message.connect(display_message)
        worker.signals.update_train_test_shape_label.connect(source.update_train_te
st_shape_label)
        worker.signals.populate_tablewidget_with_dataframe.connect(
            source.generate_qt_items_to_fill_tablewidget)

        # Початок потоку
        ui.threadpool.start(worker)
```

```
    def add_num_scaling_rule(source):
        ui = source.ui

        if source.ml_model.pre_processed_numeric_variables == []:
            display_message(QtWidgets.QMessageBox.Critical, 'Невірне введення',
                            'Числові змінні відсутні', 'Помилка')
            return

        rule_text = 'Apply Numeric Scaling'
        rule_data = {'pre_processing_action': 'apply_num_scaling'}
        item_to_add = source.create_listwidgetitem(rule_text, rule_data)
        source.add_pre_processing_rule_to_listWidget(item_to_add,
ui.preprocess_sequence_listWidget)

    def add_rm_outliers_rule(source):
        ui = source.ui

        cut_off = ui.outliers_treshold_horizontalSlider.value() / 10
        text = 'Remove Outliers, Cut-off = {}σ'.format(cut_off)
        data = {'pre_processing_action': 'rm_outliers', 'cut_off': cut_off}
        item_to_add = source.create_listwidgetitem(text, data)
        source.add_pre_processing_rule_to_listWidget(item_to_add,
ui.preprocess_sequence_listWidget)

    def add_numeric_filtering_rule(source):
        ui = source.ui
        filtering_value = ui.filtering_dataset_value_lineEdit.text()
        filtering_variable = ui.filter_columnSelection_comboBox.currentText()
        filtering_operator = ui.filter_operator_comboBox.currentText()
        if filtering_value != '':
            try:
                float(filtering_value)
            except:
                display_message(QtWidgets.QMessageBox.Critical, 'Невірне введення',
                                'Type a valid numeric input for column
{}'.format(filtering_operator), 'Помилка')
                return
            rule_text = 'Exclude values from {} {} {}'.format(filtering_variable,
filtering_operator, filtering_value)
            rule_data = {'pre_processing_action': 'apply_filtering', 'variable':
filtering_variable, 'is_numeric': True,
                         'filtering_operator': filtering_operator,
'filtering_value': filtering_value}
            item_to_add = source.create_listwidgetitem(rule_text, rule_data)
            source.add_pre_processing_rule_to_listWidget(item_to_add,
ui.preprocess_sequence_listWidget)
        else:
            display_message(QtWidgets.QMessageBox.Information, 'Empty Input',
                            'Type a valid rule', 'Помилка')
```

```
    def add_categorical_filtering_rule(source):
        ui = source.ui
        filtering_value = ui.filtering_dataset_value_comboBox.currentText()
        filtering_variable = ui.filter_columnSelection_comboBox.currentText()
        filtering_operator = ui.filter_operator_comboBox.currentText()
        rule_text = 'Exclude values from {} {} {}'.format(filtering_variable,
filtering_operator, filtering_value)
        rule_data = {'pre_processing_action': 'apply_filtering', 'variable':
filtering_variable, 'is_numeric': False,
                     'filtering_operator': filtering_operator, 'filtering_value':
filtering_value}
        item_to_add = source.create_listwidgetitem(rule_text, rule_data)
        source.add_pre_processing_rule_to_listWidget(item_to_add,
ui.preprocess_sequence_listWidget)

    def generate_filtering_rule(source):
        ui = source.ui
        ml_model = source.ml_model

        current_variable_name = ui.filter_columnSelection_comboBox.currentText()
        variable_type = ml_model.column_types_pd_series[current_variable_name].kind
        is_numeric_variable = variable_type in 'iuf'

        if is_numeric_variable:  # якщо числове значення
            source.add_numeric_filtering_rule()

        else:  # якщо не числове
            source.add_categorical_filtering_rule()

    def add_numeric_replacing_rule(source):
        ui = source.ui
        replacing_variable = ui.replace_columnSelection_comboBox.currentText()
        old_values = ui.replaced_value_lineEdit.text()
        new_values = ui.replacing_value_lineEdit.text()

        if old_values != '' and new_values != '':  # якщо вхід не порожній
            try:
                float(new_values), float(old_values)  # перевірка валідності
            except:
                display_message(QtWidgets.QMessageBox.Critical, 'Невірне
введення',  # виведення порожнього виходу
                                'Введіть числове значення для колонки
{}'.format(replacing_variable), 'Помилка')
                return
            rule_text = 'Замінити {} в {} на {}'.format(old_values,
replacing_variable, new_values)
            rule_data = {'pre_processing_action': 'replace_values', 'variable':
replacing_variable, 'is_numeric': True,
                         'old_values': old_values, 'new_values': new_values}
            item_to_add = source.create_listwidgetitem(rule_text, rule_data)
```

```
            source.add_pre_processing_rule_to_listWidget(item_to_add,
ui.preprocess_sequence_listWidget)
        else:  # Якщо вхід порожній при доданні правила
            display_message(QtWidgets.QMessageBox.Information, 'Порожнє поле',
                            'Введіть правило', 'Помилка')

    def add_categorical_replacing_rule(source):
        ui = source.ui
        replacing_variable = ui.replace_columnSelection_comboBox.currentText()
        old_values = ui.replaced_value_comboBox.currentText()
        new_values = ui.replacing_value_lineEdit.text()

        if new_values != '':  # якщо вхід не порожній
            rule_text = 'Replace {} in {} with {}'.format(old_values,
replacing_variable, new_values)
            rule_data = {'pre_processing_action': 'replace_values', 'variable':
replacing_variable, 'is_numeric': False,
                         'old_values': old_values, 'new_values': new_values}
            item_to_add = source.create_listwidgetitem(rule_text, rule_data)
            source.add_pre_processing_rule_to_listWidget(item_to_add,
ui.preprocess_sequence_listWidget)
        else:  # Якщо вхід порожній при доданні правила
            display_message(QtWidgets.QMessageBox.Information, 'Empty Input',
                            'Type a valid rule', 'Error')

    def generate_replacing_rule(source):
        ui = source.ui

        is_numeric_variable = ui.pre_process_replacing_stackedWidget.currentIndex()
== 0

        if is_numeric_variable:  # числові
            source.add_numeric_replacing_rule()

        else:  # не числові
            source.add_categorical_replacing_rule()

    def add_rm_duplicate_rows_rule(source):
        ui = source.ui

        rule_text = 'Remove Duplicate Rows'
        rule_data = {'pre_processing_action': 'rm_duplicate_rows'}
        item_to_add = source.create_listwidgetitem(rule_text, rule_data)
        source.add_pre_processing_rule_to_listWidget(item_to_add,
ui.preprocess_sequence_listWidget)

    def add_rm_constant_var_rule(source):
        ui = source.ui

        rule_text = 'Remove Constant Variables (Columns)'
```

```
        rule_data = {'pre_processing_action': 'rm_constant_var'}
        item_to_add = source.create_listwidgetitem(rule_text, rule_data)
        source.add_pre_processing_rule_to_listWidget(item_to_add,
ui.preprocess_sequence_listWidget)

    def add_pre_processing_rule_to_listWidget(source, item, listWidget):

        listWidget.addItem(item)
        source.trigger_update_pre_process_thread()

    def update_preprocess_replace_fields(source):
        ui = source.ui
        ml_model = source.ml_model
        selected_value = ui.replace_columnSelection_comboBox.currentText()

        is_numeric_variable = ml_model.column_types_pd_series[
                        selected_value].kind in 'iuf'

        if is_numeric_variable:
            ui.pre_process_replacing_stackedWidget.setCurrentIndex(0)
        else:
            ui.pre_process_replacing_stackedWidget.setCurrentIndex(1)

            ui.replaced_value_comboBox.clear()
            unique_values = ml_model.dataset[selected_value].unique().tolist()

            # Заповнення комбобоксів
            for each_value in unique_values:
                ui.replaced_value_comboBox.addItem(each_value)

    def update_preprocess_filtering_fields(source):
        ui = source.ui
        ml_model = source.ml_model

        selected_value = ui.filter_columnSelection_comboBox.currentText()
        is_numeric_variable = ml_model.column_types_pd_series[
                                selected_value].kind in 'iuf'
        if is_numeric_variable:
            ui.pre_process_filtering_stackedWidget.setCurrentIndex(0)
            if ui.filter_operator_comboBox.count() == 2:  # якщо два, то тільки ==
i !=
                ui.filter_operator_comboBox.insertItem(2, 'Greater than or equal
to')
                ui.filter_operator_comboBox.insertItem(2, 'Greater than')
                ui.filter_operator_comboBox.insertItem(2, 'Less than or equal to')
                ui.filter_operator_comboBox.insertItem(2, 'Less than')
        else:
            ui.pre_process_filtering_stackedWidget.setCurrentIndex(1)
            ui.filtering_dataset_value_comboBox.clear()
            unique_values = ml_model.dataset[selected_value].unique().tolist()
```

```
            if ui.filter_operator_comboBox.count() == 6:
                ui.filter_operator_comboBox.removeItem(2)  # Прибирання <
                ui.filter_operator_comboBox.removeItem(2)  #  <=
                ui.filter_operator_comboBox.removeItem(2)  #  >
                ui.filter_operator_comboBox.removeItem(2)  #  >=
            #Заповнення комбобоксів
            for each_value in unique_values:
                ui.filtering_dataset_value_comboBox.addItem(each_value)

    def trigger_update_pre_process_thread(source):

        ui = source.ui
        ml_model = source.ml_model

        ui.pre_process_dataset_tableWidget.spinner.start()

        worker = threads.Pre_Process_Dataset_Thread(ui, ml_model)

        worker.signals.update_pre_process_tableWidget.connect(source.generate_qt_it
ems_to_fill_tablewidget)
        worker.signals.display_message.connect(display_message)

        ui.threadpool.start(worker)

    def update_train_test_shape_label(source):
        ui = source.ui
        ml_model = source.ml_model

        dataset_shape = ml_model.pre_processed_dataset.shape

        number_of_rows_train = round(dataset_shape[0] *
ui.train_percentage_horizontalSlider.value() / 100)
        number_of_columns_train = ui.input_columns_listWidget.count()

        number_of_rows_test = round(dataset_shape[0] *
ui.test_percentage_horizontalSlider.value() / 100)
        number_of_columns_test = ui.input_columns_listWidget.count()

        ui.train_dataset_shape_label.setText('{} x {}'.format(number_of_rows_train,
number_of_columns_train))
        ui.test_dataset_shape_label.setText('{} x {}'.format(number_of_rows_test,
number_of_columns_test))

    def clear_listwidget(source, target_listwidget):
        ui = source.ui
        ml_model = source.ml_model
        is_regression = ui.regression_selection_radioButton.isChecked()

        if target_listwidget == ui.preprocess_sequence_listWidget:
            target_listwidget.clear()
```

```
            source.trigger_update_pre_process_thread()

        elif target_listwidget == ui.input_columns_listWidget:

            for _ in range(target_listwidget.count()):
                item = target_listwidget.takeItem(0)
                ui.available_columns_listWidget.addItem(item)

                # додавання змінних на вихід в комбобокс
                if item.text() in ml_model.categorical_variables:
                    ui.clas_output_colum_comboBox.addItem(item.text())

            ui.train_model_pushButton.setDisabled(True)
            source.update_train_test_shape_label()

        elif target_listwidget == ui.output_columns_listWidget:

            if is_regression:
                ui.train_model_pushButton.setDisabled(True)

            for _ in range(target_listwidget.count()):
                item = target_listwidget.takeItem(0)
                ui.available_columns_listWidget.addItem(item)

    def update_train_model_button_status(source, is_regression):
        ui = source.ui
        if is_regression:
            if ui.output_columns_listWidget.count() > 0 and
ui.input_columns_listWidget.count() > 0:
                ui.train_model_pushButton.setDisabled(False)
            else:
                ui.train_model_pushButton.setDisabled(True)
        else:
            if ui.input_columns_listWidget.count() > 0 and
ui.clas_output_colum_comboBox.count() > 0:
                ui.train_model_pushButton.setDisabled(False)
            else:
                ui.train_model_pushButton.setDisabled(True)

    def create_listwidgetitem(source, text, data):
        string_to_add = text
        my_qlist_item = QtWidgets.QListWidgetItem()
        my_qlist_item.setText(string_to_add)
        my_qlist_item.setData(QtCore.Qt.UserRole, data)
        return my_qlist_item

    def model_selection_tab_events(source):
        ui = source.ui
        is_regression = ui.regression_selection_radioButton.isChecked()
```

```
        if is_regression:
            ui.regression_and_classification_stackedWidget.setCurrentIndex(0)  #
Вкладка регресії
            ui.train_metrics_stackedWidget.setCurrentIndex(0)
            ui.output_selection_stackedWidget.setCurrentIndex(0)
            source.update_train_model_button_status(is_regression)

            if ui.nn_regression_radioButton.isChecked():
                ui.regression_parameters_stackedWidget.setCurrentIndex(0)

            elif ui.svm_regression_radioButton.isChecked():
                ui.regression_parameters_stackedWidget.setCurrentIndex(1)

            elif ui.randomforest_regression_radioButton.isChecked():
                ui.regression_parameters_stackedWidget.setCurrentIndex(2)

            elif ui.gradientboosting_regression_radioButton.isChecked():
                ui.regression_parameters_stackedWidget.setCurrentIndex(3)

        elif ui.classification_selection_radioButton.isChecked():
            ui.regression_and_classification_stackedWidget.setCurrentIndex(1)  #
Вкладка класифікації
            ui.train_metrics_stackedWidget.setCurrentIndex(1)  # Зміна на вкладку
регресії
            ui.output_selection_stackedWidget.setCurrentIndex(1)
            source.update_train_model_button_status(is_regression)

            if ui.nn_classification_radioButton.isChecked():
                ui.classification_parameters_stackedWidget.setCurrentIndex(0)

            elif ui.svm_classification_radioButton.isChecked():
                ui.classification_parameters_stackedWidget.setCurrentIndex(1)

            elif ui.randomforest_classification_radioButton.isChecked():
                ui.classification_parameters_stackedWidget.setCurrentIndex(2)

            elif ui.gradientboosting_classification_radioButton.isChecked():
                ui.classification_parameters_stackedWidget.setCurrentIndex(3)

            elif ui.knn_classification_radioButton.isChecked():
                ui.classification_parameters_stackedWidget.setCurrentIndex(4)

    def check_neurons_number(source,widget):
        row = widget.currentRow()
        column = widget.currentColumn()

        widget.blockSignals(True)
        item = QtWidgets.QTableWidgetItem()
        item.setTextAlignment(QtCore.Qt.AlignCenter)
        try:
```

```
            integer_value = int(widget.item(row,column).text())
            if integer_value <= 0:
                display_message(QtWidgets.QMessageBox.Information, 'Invalid Input',
                                'The number of neurons must be an integer greater
than 0', 'Error')
                item.setText('1')
                widget.setItem(row, column,item)
        except:
            display_message(QtWidgets.QMessageBox.Critical, 'Invalid Input',
                            'The number of neurons must be an integer greater than
0', 'Error')
            item.setText('1')
            widget.setItem(row, column, item)
        widget.blockSignals(False)

    def generate_qt_items_to_fill_tablewidget(source, table_widget,
filling_dataframe):

        table_widget.clear()

        # Заповеннння віджету таблиці даних з набору
        number_of_rows_to_display = 50
        table_widget.setRowCount(len(filling_dataframe.head(number_of_rows_to_displ
ay)))
        table_widget.setColumnCount(len(filling_dataframe.columns))

        # Назви стовпців
        data = {'header_labels': filling_dataframe.columns}
        # оновлення потоку з віджетом
        source.update_table_widget(table_widget, 'update_header', data)

        # заповнення таблиці
        for i in range(table_widget.rowCount()):
            for j in range(table_widget.columnCount()):
                dataset_value = filling_dataframe.iloc[i, j]
                dataset_value_converted = dataset_value if (type(dataset_value) is
str) else '{:}'.format(dataset_value)
                qt_item = QtWidgets.QTableWidgetItem(dataset_value_converted)
                qt_item.setTextAlignment(QtCore.Qt.AlignHCenter)
                # оновлення потоку з віджетом
                data = {'i': i, 'j': j, 'qt_item': qt_item}
                source.update_table_widget(table_widget, 'fill_table', data)
        source.update_table_widget(table_widget, 'stop_spinner', data)

    def update_visualisation_widgets(source):
        ui = source.ui
        ml_model = source.ml_model

        selected_column = ui.variable_to_plot_comboBox.currentText()
```

```
        ui.columnSummary_textBrowser.clear()
        description = ml_model.dataset[selected_column].describe()
        for i in range(len(description)):
            ui.columnSummary_textBrowser.append('{} =
{}'.format(description.keys()[i].title(), description.values[i]))

        is_categorical = ml_model.column_types_pd_series[
                             selected_column].kind not in 'iuf'

        source.trigger_plot_matplotlib_to_qt_widget_thread(target_widget=ui.dataVis
ualisePlot_widget,
                                                           content={'data':
ml_model.dataset[selected_column],
                                                                    'is_categorical':
is_categorical})

    def update_visualisation_options(source):
        ui = source.ui
        ml_model = source.ml_model

        selected_column = ui.variable_to_plot_comboBox.currentText()
        radio_buttons_list = [ui.plot_radioButton, ui.boxplot_radioButton,
ui.histogram_radioButton]


        if ml_model.column_types_pd_series[selected_column].kind in 'iuf':
            radio_buttons_list[0].setEnabled(True)
            radio_buttons_list[1].setEnabled(True)
            radio_buttons_list[2].setEnabled(True)

            checked_objects = list(map(lambda x: x.isChecked(),
radio_buttons_list))
            if not any(checked_objects):
                radio_buttons_list[0].setChecked(True)  #Якщо жодна не відмічена,
відмітити першу

        else:  # вимкнення іншої візуалізації для нечислових
            radio_buttons_list[0].setEnabled(False)
            radio_buttons_list[1].setEnabled(False)
            radio_buttons_list[2].setEnabled(False)

        source.update_visualisation_widgets()

    def trigger_plot_matplotlib_to_qt_widget_thread(source, target_widget,
content):

        worker = threads.Plotting_in_MplWidget_Thread(source.ui, target_widget,
content)

        source.ui.threadpool.start(worker)
```

```
    def update_table_widget(source, table_widget, function, data):
        ui = source.ui

        if function == 'update_header':
            table_widget.setHorizontalHeaderLabels(data['header_labels'])
            header = table_widget.horizontalHeader()
            header.setSectionResizeMode(QtWidgets.QHeaderView.ResizeToContents)

        elif function == 'fill_table':
            i = data['i']
            j = data['j']
            qt_item = data['qt_item']
            table_widget.setItem(i, j, qt_item)

        elif function == 'stop_spinner':
            table_widget.spinner.stop()
            if table_widget.objectName() == 'dataset_tableWidget':
                ui.load_file_pushButton.setDisabled(False)
                ui.example_dataset_comboBox.setDisabled(False)
            elif table_widget.objectName() == 'pre_process_dataset_tableWidget':
                source.update_train_test_shape_label()

    def update_nn_layers_table(source, table, value):
        # blockSignals не дає виконати check_neurons_number звідси
        table.blockSignals(True)
        if value > table.rowCount():
            while value > table.rowCount():
                table.insertRow(table.rowCount())
                item = QtWidgets.QTableWidgetItem(str(10))
                item.setTextAlignment(QtCore.Qt.AlignCenter)
                table.setItem(table.rowCount() - 1, 0, item)
                item = QtWidgets.QTableWidgetItem('Hidden Layer ' +
str(table.rowCount()))
                table.setVerticalHeaderItem(table.rowCount() - 1, item)
        else:
            while value < table.rowCount():
                table.removeRow(table.rowCount() - 1)
        table.blockSignals(False)

    def update_svm_model_parameters(source, action, is_regression):

        ui = source.ui

        if is_regression:
            combobox = ui.reg_svm_kernel_comboBox
            spin_box = ui.reg_svm_kernel_degree_spinBox
            c_slider = ui.reg_svm_C_horizontalSlider
            c_label = ui.reg_svm_C_label
            epsilon_slider = ui.reg_svm_episilon_horizontalSlider
```

```
            epsilon_label = ui.reg_svm_episilon_label
            max_iter_slider = ui.reg_svm_maxiter_horizontalSlider
            max_iter_label = ui.reg_svm_maxiter_label
            check_box = ui.reg_svm_maxiter_nolimit_checkBox
        else:
            combobox = ui.clas_svm_kernel_comboBox
            spin_box = ui.clas_svm_kernel_degree_spinBox
            c_slider = ui.clas_svm_C_horizontalSlider
            c_label = ui.clas_svm_C_label
            max_iter_slider = ui.clas_svm_maxiter_horizontalSlider
            max_iter_label = ui.clas_svm_maxiter_label
            check_box = ui.clas_svm_maxiter_nolimit_checkBox

        if action == 'kernel_change':
            if combobox.currentText() == 'poly':
                spin_box.setEnabled(True)
            else:
                spin_box.setEnabled(False)
        elif action == 'regularisation_change':
            source.update_label_from_slider_change(c_slider.value(),c_label)
        elif action == 'epsilon_change':
            source.update_label_from_slider_change(epsilon_slider.value(),epsilon_l
abel)
        elif action == 'max_iter_change':
            source.update_label_from_slider_change(max_iter_slider.value(),max_iter
_label)
        elif action == 'no_limit_click':
            if check_box.isChecked():
                max_iter_slider.setEnabled(False)
            else:
                max_iter_slider.setEnabled(True)

    def display_training_results(source, result, model_parameters):
        ui = source.ui
        ui.spinner_traning_results.stop()
        ui.train_model_pushButton.setDisabled(False)

        if model_parameters['is_regression']:
            ui.reg_mse_label.setText('{:.4f}'.format(result['mse']))
            ui.reg_rmse_label.setText('{:.4f}'.format(result['rmse']))
            ui.reg_r2_label.setText('{:.4f}'.format(result['r2_score']))
            ui.reg_mea_label.setText('{:.4f}'.format(result['mae']))
            source.trigger_plot_matplotlib_to_qt_widget_thread(target_widget=ui.mod
el_train_widget,
                                                               content={'data':
result['data_to_plot'],
                                                                        'output_varia
bles': model_parameters[
                                                                            'output_v
ariables'],
```

```
                                                                'is_regressio
n': model_parameters[
                                                                    'is_regre
ssion']})

        else:
            ui.clas_accuracy_label.setText('{:.4f}'.format(result['accuracy']))
            ui.clas_recall_label.setText('{:.4f}'.format(result['recall_score']))
            ui.clas_precision_label.setText('{:.4f}'.format(result['precision_score
']))
            ui.clas_f1_score_label.setText('{:.4f}'.format(result['f1_score']))
            source.trigger_plot_matplotlib_to_qt_widget_thread(target_widget=ui.mod
el_train_widget,
                                                               content={'data':
result['data_to_plot'],
                                                                'output_varia
bles': model_parameters[
                                                                    'output_v
ariables'],
                                                                'is_regressio
n': model_parameters[
                                                                    'is_regre
ssion']})

    def update_input_output_columns(source, target_object):
        ui = source.ui
        ml_model = source.ml_model
        is_regression = ui.regression_selection_radioButton.isChecked()

        if target_object == 'clear_output_variables':
            if ui.tabs_widget.currentIndex() == 4 and not is_regression:
                ui.clear_output_columns_pushButton.click()
            return

        for selected_item in ui.available_columns_listWidget.selectedItems():
            item =
ui.available_columns_listWidget.takeItem(ui.available_columns_listWidget.row(select
ed_item))
            is_variable_categorical = selected_item.text() in
ml_model.categorical_variables
            is_output_variable = target_object.objectName() ==
'output_columns_listWidget'
            if is_regression and is_variable_categorical and is_output_variable:
                ui.available_columns_listWidget.addItem(item)
                display_message(QtWidgets.QMessageBox.Information, 'Invalid Input',
                                'Categorical variables should not be used as
regression output', 'Error')
            else:
                target_object.addItem(item)
                combobox = ui.clas_output_colum_comboBox
```

```
                items_in_combobox = [combobox.itemText(i) for i in
range(combobox.count())]
                if selected_item.text() in items_in_combobox:
                    item_index = items_in_combobox.index(selected_item.text())
                    combobox.removeItem(item_index)

        if target_object.objectName() == 'input_columns_listWidget':
            source.update_train_test_shape_label()

        source.update_train_model_button_status(is_regression)

    def update_label_from_slider_change(source, slider_value, label_object):
        ui = source.ui
        ml_model = source.ml_model

        label_object.setText('{}'.format(slider_value))

        if label_object.objectName() == 'reg_nn_layers_label':
            source.update_nn_layers_table(ui.reg_nn_layers_tableWidget,
slider_value)
        elif label_object.objectName() == 'clas_nn_layers_label':
            source.update_nn_layers_table(ui.clas_nn_layers_tableWidget,
slider_value)
        elif label_object.objectName() == 'outliers_treshold_label':
            label_object.setText('{:.1f}'.format(slider_value / 10))
        elif label_object.objectName() == 'reg_nn_val_percent_label':
            label_object.setText('{}%'.format(slider_value))
        elif label_object.objectName() == 'reg_nn_alpha_label':
            label_object.setText('{}'.format(slider_value / 10000))
        elif label_object.objectName() == 'clas_nn_val_percent_label':
            label_object.setText('{}%'.format(slider_value))
        elif label_object.objectName() == 'clas_nn_alpha_label':
            label_object.setText('{}'.format(slider_value / 10000))
        elif label_object.objectName() == 'train_percentage_label':
            label_object.setText('{}%'.format(slider_value))
            ui.test_percentage_horizontalSlider.setValue(100 - slider_value)
            if ml_model.is_data_loaded:
                source.update_train_test_shape_label()
        elif label_object.objectName() == 'test_percentage_label':
            label_object.setText('{}%'.format(slider_value))
            ui.train_percentage_horizontalSlider.setValue(100 - slider_value)
            if ml_model.is_data_loaded:
                source.update_train_test_shape_label()
        elif label_object.objectName() == 'clas_svm_C_label':
            label_object.setText('{:.1f}'.format(slider_value / 10))

    def trigger_train_model_thread(source):
        ui = source.ui
        ml_model = source.ml_model
```

```
        train_percentage = (ui.train_percentage_horizontalSlider.value() / 100)
        test_percentage = (ui.test_percentage_horizontalSlider.value() / 100)
        shuffle_samples = ui.shuffle_samples_checkBox.isChecked()
        model_parameters = {'train_percentage': train_percentage,
'test_percentage': test_percentage,
                            'shuffle_samples': shuffle_samples}
        is_regression = ui.regression_selection_radioButton.isChecked()

        input_variables = []
        for i in range(ui.input_columns_listWidget.count()):
            input_variables.append(ui.input_columns_listWidget.item(i).text())

        if is_regression:
            output_variables = []
            for i in range(ui.output_columns_listWidget.count()):
                output_variables.append(ui.output_columns_listWidget.item(i).text()
)
        else:
            output_variables = [ui.clas_output_colum_comboBox.currentText()]

        if is_regression:
            algorithm_index = [ui.nn_regression_radioButton.isChecked(),
ui.svm_regression_radioButton.isChecked(),
                               ui.randomforest_regression_radioButton.isChecked(),
                               ui.gradientboosting_regression_radioButton.isChecked
()].index(1)
            algorithm = ['nn', 'svm', 'random_forest',
'grad_boosting'][algorithm_index]

            if algorithm == 'nn':
                n_of_hidden_layers = ui.reg_nn_layers_horizontalSlider.value()
                n_of_neurons_each_layer = []
                for i in range(n_of_hidden_layers):
                    n_of_neurons_each_layer.append(int(ui.reg_nn_layers_tableWidget
.item(i, 0).text()))
                activation_func = ui.reg_nn_actvfunc_comboBox.currentText()
                solver = ui.reg_nn_solver_comboBox.currentText()
                learning_rate = ui.reg_nn_learnrate_comboBox.currentText()
                max_iter = ui.reg_nn_max_iter_horizontalSlider.value()
                alpha = ui.reg_nn_alpha_horizontalSlider.value() / 10000
                validation_percentage =
ui.reg_nn_val_percentage_horizontalSlider.value() / 100

                algorithm_parameters = {'n_of_hidden_layers': n_of_hidden_layers,
                                        'n_of_neurons_each_layer':
n_of_neurons_each_layer,
                                        'activation_func': activation_func,
'solver': solver,
                                        'learning_rate': learning_rate,
                                        'max_iter': max_iter, 'alpha': alpha,
```

```
                                        'validation_percentage':
validation_percentage}
            elif algorithm == 'svm':
                kernel = ui.reg_svm_kernel_comboBox.currentText()
                kernel_degree = ui.reg_svm_kernel_degree_spinBox.value()
                regularisation_parameter = float(ui.reg_svm_C_label.text())
                is_shrinking_enables =  ui.reg_svm_shirinking_checkBox.isChecked()
                epsilon = float(ui.reg_svm_episilon_label.text())
                max_iter_no_limit_checked =
ui.reg_svm_maxiter_nolimit_checkBox.isChecked()
                max_iter = int(ui.reg_svm_maxiter_label.text())
                algorithm_parameters = {'kernel': kernel,
                                        'kernel_degree': kernel_degree,
                                        'regularisation_parameter':
regularisation_parameter,
                                        'is_shrinking_enables':
is_shrinking_enables,
                                        'epsilon': epsilon,
                                        'max_iter_no_limit_checked':
max_iter_no_limit_checked,
                                        'max_iter': max_iter}
            elif algorithm == 'random_forest':
                algorithm_parameters = {}
            elif algorithm == 'grad_boosting':
                algorithm_parameters = {}
        else:
            algorithm_index = [ui.nn_classification_radioButton.isChecked(),
                               ui.svm_classification_radioButton.isChecked(),
                               ui.randomforest_classification_radioButton.isChecked
(),
                               ui.gradientboosting_classification_radioButton.isChe
cked(),
                               ui.knn_classification_radioButton.isChecked()].index
(1)
            algorithm = ['nn', 'svm', 'random_forest', 'grad_boosting',
'knn'][algorithm_index]

            if algorithm == 'nn':
                n_of_hidden_layers = ui.clas_nn_layers_horizontalSlider.value()
                n_of_neurons_each_layer = []
                for i in range(n_of_hidden_layers):
                    n_of_neurons_each_layer.append(int(ui.clas_nn_layers_tableWidge
t.item(i, 0).text()))
                activation_func = ui.clas_nn_actvfunc_comboBox.currentText()
                solver = ui.clas_nn_solver_comboBox.currentText()
                learning_rate = ui.clas_nn_learnrate_comboBox.currentText()
                max_iter = ui.clas_nn_max_iter_horizontalSlider.value()
                alpha = ui.clas_nn_alpha_horizontalSlider.value() / 10000
                validation_percentage =
ui.clas_nn_val_percentage_horizontalSlider.value() / 100
```

```
                algorithm_parameters = {'n_of_hidden_layers': n_of_hidden_layers,
                                        'n_of_neurons_each_layer':
n_of_neurons_each_layer,
                                        'activation_func': activation_func,
'solver': solver,
                                        'learning_rate': learning_rate,
                                        'max_iter': max_iter, 'alpha': alpha,
                                        'validation_percentage':
validation_percentage}
            elif algorithm == 'svm':
                kernel = ui.clas_svm_kernel_comboBox.currentText()
                kernel_degree = ui.clas_svm_kernel_degree_spinBox.value()
                regularisation_parameter = float(ui.clas_svm_C_label.text())
                is_shrinking_enables =  ui.clas_svm_shirinking_checkBox.isChecked()
                max_iter_no_limit_checked =
ui.clas_svm_maxiter_nolimit_checkBox.isChecked()
                max_iter = int(ui.clas_svm_maxiter_label.text())
                algorithm_parameters = {'kernel': kernel,
                                        'kernel_degree': kernel_degree,
                                        'regularisation_parameter':
regularisation_parameter,
                                        'is_shrinking_enables':
is_shrinking_enables,
                                        'max_iter_no_limit_checked':
max_iter_no_limit_checked,
                                        'max_iter': max_iter}
            elif algorithm == 'random_forest':
                algorithm_parameters = {}
            elif algorithm == 'grad_boosting':
                algorithm_parameters = {}
            elif algorithm == 'knn':
                algorithm_parameters = {}

        model_parameters.update(
            {'is_regression': is_regression, 'algorithm': algorithm,
'input_variables': input_variables,
             'output_variables': output_variables})

        worker = threads.Train_Model_Thread(ml_model, model_parameters,
algorithm_parameters, ui)

        worker.signals.finished.connect(source.display_training_results)

        ui.train_model_pushButton.setDisabled(True)
        ui.spinner_traning_results.start()

        # Запуск на окремому від інтерфейсу потоці
        ui.threadpool.start(worker)
```

```
    def remove_item_from_listwidget(source, target_listwidget):
        ui = source.ui
        ml_model = source.ml_model

        for item in target_listwidget.selectedItems():
            taken_item = target_listwidget.takeItem(target_listwidget.row(item))

            if target_listwidget == ui.input_columns_listWidget or
target_listwidget == ui.output_columns_listWidget:
                    ui.available_columns_listWidget.addItem(taken_item)

                    # Додавання змінних в вихідний клас
                    if item.text() in ml_model.categorical_variables and
target_listwidget == ui.input_columns_listWidget:
                        target_listwidget.addItem(taken_item.text())

        if target_listwidget == ui.input_columns_listWidget or target_listwidget ==
ui.output_columns_listWidget:
            source.update_train_model_button_status(ui.regression_selection_radioBu
tton.isChecked())

        if target_listwidget == ui.preprocess_sequence_listWidget:
            source.trigger_update_pre_process_thread()

def transform_to_resource_path(relative_path):
    """Get absolute path to resource, works for dev and for PyInstaller.

    Args:
        relative_path (str): path to the file.

    Returns:
        str: path that works for both debug and standalone app
    """
    try:
        base_path = sys._MEIPASS
    except Exception:
        base_path = abspath(".")

    return join(base_path, relative_path)

def display_message(icon, main_message, informative_message, window_title):
    msg = QtWidgets.QMessageBox()
    msg.setIcon(icon)
    msg.setText(main_message)
    msg.setInformativeText(informative_message)
    msg.setWindowTitle(window_title)
    msg.exec()

def get_project_root_directory():
    full_path = os.path.realpath(__file__)
```

```
    path, filename = os.path.split(full_path)
    root_directory = path + '/../'
    return root_directory
```

Файл main.py

```
import model as md
import sklearn
from view import QtCore, QtWidgets, Ui_MainWindow
from controller import ViewController
import controller
import sys

app = QtCore.QCoreApplication.instance()
if app is None:
    app = QtWidgets.QApplication(sys.argv)
MainWindow = QtWidgets.QMainWindow()
ui = Ui_MainWindow()
ui.setupUi(MainWindow)
user_ML_model = md.MlModel()
view_controller = ViewController(ui,user_ML_model)
MainWindow.show()
sys.exit(app.exec_())
```

Файл model.py

```
from sklearn import svm
from sklearn.svm import SVR
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.multioutput import MultiOutputRegressor
from sklearn.neural_network import MLPRegressor, MLPClassifier
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error,
recall_score, f1_score, precision_score, \
    accuracy_score, confusion_matrix

import operator
import pandas as pd
import numpy as np
import os
import joblib

class MlModel:

    def __init__(source):
```

```
        source.dataset = pd.DataFrame()
        source.column_types_pd_series = []
        source.categorical_variables = []
        source.integer_variables = []
        source.numeric_variables = []

        source.pre_processed_dataset = pd.DataFrame()
        source.pre_processed_column_types_pd_series = []
        source.pre_processed_categorical_variables = []
        source.pre_processed_integer_variables = []
        source.pre_processed_numeric_variables = []

        source.is_data_loaded = False
        source.input_scaler = []

    def read_dataset(source, address):

        filename, file_extension = os.path.splitext(address)

        try:
            if file_extension == '.csv':
                source.dataset = pd.read_csv(address)
            elif file_extension == '.xls' or file_extension == '.xlsx':
                source.dataset = pd.read_excel(address)
            else:
                return 'invalidExtension'
            source.is_data_loaded = True
            source.dataset.dropna(inplace = True)
            source.pre_processed_dataset = source.dataset.copy()
            source.update_datasets_info()
            return 'success'
        except:
            return 'inputFileException'

    def update_datasets_info(source):

        dataSet = source.dataset
        dataSet.reset_index(inplace=True, drop=True)
        source.column_types_pd_series = dataSet.dtypes
        source.categorical_variables =
dataSet.select_dtypes(include=['object']).columns.to_list()
        source.integer_variables =
dataSet.select_dtypes(include=['int64']).columns.to_list()
        source.numeric_variables = dataSet.select_dtypes(include=['int64',
'float64']).columns.to_list()

        dataSet = source.pre_processed_dataset
        dataSet.reset_index(inplace=True, drop=True)
        source.pre_processed_column_types_pd_series = dataSet.dtypes
```

```
        source.pre_processed_categorical_variables =
dataSet.select_dtypes(include=['object']).columns.to_list()
        source.pre_processed_integer_variables =
dataSet.select_dtypes(include=['int64']).columns.to_list()
        source.pre_processed_numeric_variables =
dataSet.select_dtypes(include=['int64', 'float64']).columns.to_list()

    def remove_outliers(source, cut_off):
        # Видалення рядів з виключеннями
        numeric_columns =
source.pre_processed_dataset.select_dtypes(include=['float64',
'int']).columns.to_list()
        z_score = stats.zscore(source.pre_processed_dataset[numeric_columns])
        source.pre_processed_dataset =
source.pre_processed_dataset[(np.abs(z_score) < cut_off).all(axis=1)]
        source.update_datasets_info()

    def scale_numeric_values(source):

        dataset = source.pre_processed_dataset
        source.input_scaler = MinMaxScaler(feature_range=(-1, 1))
        standardised_numeric_input =
source.input_scaler.fit_transform(dataset[source.pre_processed_numeric_variables])
        # Оновлення змінених даних в наборі
        dataset[source.pre_processed_numeric_variables] =
standardised_numeric_input
        source.update_datasets_info()

    def remove_duplicate_rows(source):
        source.pre_processed_dataset.drop_duplicates(inplace=True)
        source.update_datasets_info()

    def remove_constant_variables(source):
        dataset = source.pre_processed_dataset
        source.pre_processed_dataset = dataset.loc[:, (dataset !=
dataset.iloc[0]).any()]
        source.update_datasets_info()

    def replace_values(source, target_variable, new_value, old_values):

        variable_data_type =
source.pre_processed_column_types_pd_series[target_variable]

        if source.pre_processed_column_types_pd_series[target_variable].kind ==
'f':
            value_to_replace = float(old_values)
            new_value = float(new_value)
        elif source.pre_processed_column_types_pd_series[target_variable].kind ==
'i':
            value_to_replace = int(old_values)
```

```
            new_value = int(new_value)
        else:
            value_to_replace = old_values
        value_to_replace =
pd.Series(value_to_replace).astype(variable_data_type).values[0]
        # Конвертування в float або int якщо були дроби в string
        source.pre_processed_dataset[target_variable].replace(to_replace=value_to_r
eplace, value=new_value,inplace=True)

        source.update_datasets_info()

    def mean_absolute_percentage_error(source,y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        subtraction = (y_true - y_pred)
        abs_pct_error = np.abs(np.divide(subtraction, y_true,
out=np.zeros_like(subtraction), where=y_true != 0))
        return (abs_pct_error * 100)

    def filter_out_values(source, filtering_variable, filtering_value,
filtering_operator):

        column_of_filtering_variable =
source.pre_processed_dataset[filtering_variable]
        dataset = source.pre_processed_dataset
        if source.pre_processed_column_types_pd_series[filtering_variable].kind ==
'f':
            filtering_value = float(filtering_value)
        if source.pre_processed_column_types_pd_series[filtering_variable].kind ==
'i':
            filtering_value = int(filtering_value)
        if filtering_operator == 'Equal to':
            source.pre_processed_dataset =
dataset[~operator.eq(column_of_filtering_variable, filtering_value)]
        elif filtering_operator == 'Not equal to':
            source.pre_processed_dataset =
dataset[~operator.ne(column_of_filtering_variable, filtering_value)]
        elif filtering_operator == 'Less than':
            source.pre_processed_dataset =
dataset[~operator.lt(column_of_filtering_variable, filtering_value)]
        elif filtering_operator == 'Less than or equal to':
            source.pre_processed_dataset
=dataset[~operator.le(column_of_filtering_variable, filtering_value)]
        elif filtering_operator == 'Greater than':
            source.pre_processed_dataset =
dataset[~operator.gt(column_of_filtering_variable, filtering_value)]
        elif filtering_operator == 'Greater than or equal to':
            source.pre_processed_dataset
=dataset[~operator.ge(column_of_filtering_variable, filtering_value)]

        source.update_datasets_info()
```

```
    def split_data_train_test(source, model_parameters):

        # Копіювання набору данних для відділення
        input_dataset = source.pre_processed_dataset[
            model_parameters['input_variables'] +
model_parameters['output_variables']].copy()
        input_dataset.reset_index(inplace=True)
        categorical_variables_in_training =
list(set(source.pre_processed_categorical_variables) & set(
            model_parameters['input_variables']))

        source.categorical_encoders = {}
        encoded_categorical_columns = pd.DataFrame()
        for column in categorical_variables_in_training:
            source.categorical_encoders[column] = LabelEncoder()
            values_to_fit_transform = input_dataset[column].values
            source.categorical_encoders[column].fit(values_to_fit_transform)
            encoded_categorical_columns[column] =
source.categorical_encoders[column].transform(values_to_fit_transform)

        dataIndexes = np.array(input_dataset.index)
        if model_parameters['shuffle_samples']:
            np.random.shuffle(dataIndexes)

        # Розділення індексів
        trainIndexes = dataIndexes[0:round(len(dataIndexes) *
model_parameters['train_percentage'])]
        testIndexes = dataIndexes[round(len(dataIndexes) *
model_parameters['train_percentage']):]

        # Заміна категоричних на закодовані значення
        input_dataset[categorical_variables_in_training] =
encoded_categorical_columns

        trainDataset = input_dataset.loc[trainIndexes]
        testDataset = input_dataset.loc[testIndexes]

        # Конвертування в numpy масиви
        x_train = trainDataset[model_parameters['input_variables']].values
        x_test = testDataset[model_parameters['input_variables']].values
        y_train = trainDataset[model_parameters['output_variables']].values
        y_test = testDataset[model_parameters['output_variables']].values

        if len(model_parameters['output_variables']) == 1:
            y_train = y_train.ravel()
            y_test = y_test.ravel()
```

```
        if not model_parameters['is_regression'] and
source.pre_processed_column_types_pd_series[
            model_parameters['output_variables'][0]].kind == 'i':
            original_target_categories =
source.dataset[model_parameters['output_variables']].values
            y_train = original_target_categories[trainIndexes]
            y_test = original_target_categories[testIndexes]

        return {'x_train': x_train, 'x_test': x_test, 'y_train': y_train, 'y_test':
y_test}

    def train(source, model_parameters, algorithm_parameters):

        split_dataset = source.split_data_train_test(model_parameters)
        x_train = split_dataset['x_train']
        x_test = split_dataset['x_test']
        y_train = split_dataset['y_train']
        y_test = split_dataset['y_test']

        if model_parameters['is_regression']:

            if model_parameters['algorithm'] == 'nn':
                ml_model =
MLPRegressor(hidden_layer_sizes=tuple(algorithm_parameters['n_of_neurons_each_layer
']),
                                             max_iter=algorithm_parameters['max_iter'],
                                             solver=algorithm_parameters['solver'],
                                             activation=algorithm_parameters['activation
_func'],
                                             alpha=algorithm_parameters['alpha'],
                                             learning_rate=algorithm_parameters['learnin
g_rate'],
                                             validation_fraction=algorithm_parameters['v
alidation_percentage'])
                ml_model.fit(x_train, y_train)
                y_pred = ml_model.predict(x_test)
            elif model_parameters['algorithm'] == 'svm':
                max_iter_no_limit_checked =
algorithm_parameters['max_iter_no_limit_checked']
                if max_iter_no_limit_checked:
                    svm_max_iter = -1
                else:
                    svm_max_iter = algorithm_parameters['max_iter']
                ml_model = SVR(kernel=algorithm_parameters['kernel'],
                               degree=algorithm_parameters['kernel_degree'],
                               C=algorithm_parameters['regularisation_parameter']
,
                               shrinking=algorithm_parameters['is_shrinking_enabl
es'],
                               epsilon=algorithm_parameters['epsilon'],
```

```
                                    max_iter=svm_max_iter)
                if len(y_train.shape)>1:
                    ml_model = MultiOutputRegressor(ml_model)
                ml_model.fit(x_train, y_train)
                y_pred = ml_model.predict(x_test)
            elif model_parameters['algorithm'] == 'random_forest':
                algorithm_parameters = []
            elif model_parameters['algorithm'] == 'grad_boosting':
                algorithm_parameters = []

            r2_score_result = r2_score(y_test, y_pred)
            mse = mean_squared_error(y_test, y_pred)
            mae = mean_absolute_error(y_test, y_pred)
            rmse = mean_squared_error(y_test, y_pred, squared=False)

            percentage_errors = source.mean_absolute_percentage_error(y_test,
y_pred)
            if len(model_parameters['output_variables']) == 1:
                data_to_plot = percentage_errors
            else:
                data_to_plot = {'labels': model_parameters['output_variables'],
'values':percentage_errors.mean(axis=0)}

            training_output = {'r2_score': r2_score_result, 'mse': mse, 'mae': mae,
'rmse': rmse,
                               'data_to_plot': data_to_plot}

            joblib.dump(ml_model, "model.pkl")
            return training_output

        else:
            source.output_class_label_encoder = LabelEncoder()
            source.output_class_label_encoder.fit(np.concatenate((y_train,
y_test)).ravel())
            encoded_y_train =
source.output_class_label_encoder.transform(y_train.ravel())
            encoded_y_test =
source.output_class_label_encoder.transform(y_test.ravel())

            if model_parameters['algorithm'] == 'nn':
                ml_model =
MLPClassifier(hidden_layer_sizes=tuple(algorithm_parameters['n_of_neurons_each_laye
r']),
                                          max_iter=algorithm_parameters['max_iter'],
                                          solver=algorithm_parameters['solver'],
                                          activation=algorithm_parameters['activatio
n_func'],
                                          alpha=algorithm_parameters['alpha'],
                                          learning_rate=algorithm_parameters['learni
ng_rate'],
```

```
                                        validation_fraction=algorithm_parameters['
validation_percentage'])
                ml_model.fit(x_train, encoded_y_train)
                encoded_y_pred = ml_model.predict(x_test)
            elif model_parameters['algorithm'] == 'svm':
                max_iter_no_limit_checked =
algorithm_parameters['max_iter_no_limit_checked']
                if max_iter_no_limit_checked:
                    svm_max_iter = -1
                else:
                    svm_max_iter = algorithm_parameters['max_iter']
                ml_model = svm.SVC(kernel=algorithm_parameters['kernel'],
                                   degree=algorithm_parameters['kernel_degree'],
                                   C=algorithm_parameters['regularisation_parameter']
,
                                   shrinking=algorithm_parameters['is_shrinking_enabl
es'],
                                   max_iter=svm_max_iter)
                ml_model.fit(x_train, encoded_y_train)
                encoded_y_pred = ml_model.predict(x_test)
            elif model_parameters['algorithm'] == 'random_forest':
                algorithm_parameters = []
            elif model_parameters['algorithm'] == 'grad_boosting':
                algorithm_parameters = []
            elif model_parameters['algorithm'] == 'knn':
                algorithm_parameters = []

            number_of_classes = len(np.unique(np.concatenate((y_train, y_test))))
            if number_of_classes > 2:
                average_value = 'macro'
            else:
                average_value = 'binary'

            recall = recall_score(encoded_y_test, encoded_y_pred,
average=average_value, zero_division=0)
            f1 = f1_score(encoded_y_test, encoded_y_pred, average=average_value,
zero_division=0)
            accuracy = accuracy_score(encoded_y_test, encoded_y_pred)
            precision = precision_score(encoded_y_test, encoded_y_pred,
average=average_value, zero_division=0)

            df_conf = pd.DataFrame(confusion_matrix(encoded_y_test,
encoded_y_pred))
            df_conf.set_index(source.output_class_label_encoder.inverse_transform(d
f_conf.index), inplace=True)
            df_conf.columns =
source.output_class_label_encoder.inverse_transform(df_conf.columns)

            training_output = {'recall_score': recall, 'f1_score': f1,
'precision_score': precision,
```

```
                            'accuracy': accuracy,
                            'data_to_plot': df_conf}
            joblib.dump(ml_model, "model.pkl")
            return training_output
```

Файл widget.py

```
from PyQt5 import QtWidgets, QtGui, QtCore
from matplotlib.backends.backend_qt5agg import FigureCanvas
from matplotlib.figure import Figure
import math

#QtWidgets.QWidget використовується щоб виводити графіки з matplotlib і Pandas в Qt
віджет
class MplWidget(QtWidgets.QWidget):

    def __init__(source, parent=None):
        QtWidgets.QWidget.__init__(source, parent)

        source.canvas = FigureCanvas(Figure())

        vertical_layout = QtWidgets.QVBoxLayout()
        vertical_layout.addWidget(source.canvas)

        source.canvas.axes = source.canvas.figure.add_subplot(111)
        source.canvas.figure.set_tight_layout(True)
        source.setLayout(vertical_layout)
        source.canvas.axes.axis('off')  # Вимкнення списків для ініціалізації


class QtWaitingSpinner(QtWidgets.QWidget):
    def __init__(source, parent, centerOnParent=True,
disableParentWhenSpinning=False, modality=QtCore.Qt.NonModal):
        super().__init__(parent)

        source._centerOnParent = centerOnParent
        source._disableParentWhenSpinning = disableParentWhenSpinning

        source._color = QtGui.QColor(QtCore.Qt.black)
        source._roundness = 100.0
        source._minimumTrailOpacity = 3.14159265358979323846
        source._trailFadePercentage = 80.0
        source._revolutionsPerSecond = 1.57079632679489661923
        source._numberOfLines = 20
        source._lineLength = 10
        source._lineWidth = 2
        source._innerRadius = 10
        source._currentCounter = 0
```

```
        source._isSpinning = False

        source._timer = QtCore.QTimer(source)
        source._timer.timeout.connect(source.rotate)
        source.updateSize()
        source.updateTimer()
        source.hide()

        source.setWindowModality(modality)
        source.setAttribute(QtCore.Qt.WA_TranslucentBackground)

    def paintEvent(source, QPaintEvent):
        source.updatePosition()
        painter = QtGui.QPainter(source)
        painter.fillRect(source.rect(), QtCore.Qt.transparent)
        painter.setRenderHint(QtGui.QPainter.Antialiasing, True)

        if source._currentCounter >= source._numberOfLines:
            source._currentCounter = 0

        painter.setPen(QtCore.Qt.NoPen)
        for i in range(0, source._numberOfLines):
            painter.save()
            painter.translate(source._innerRadius + source._lineLength,
source._innerRadius + source._lineLength)
            rotateAngle = float(360 * i) / float(source._numberOfLines)
            painter.rotate(rotateAngle)
            painter.translate(source._innerRadius, 0)
            distance = source.lineCountDistanceFromPrimary(i,
source._currentCounter, source._numberOfLines)
            color = source.currentLineColor(distance, source._numberOfLines,
source._trailFadePercentage,
                                            source._minimumTrailOpacity,
source._color)
            painter.setBrush(color)
            painter.drawRoundedRect(QtCore.QRect(0, int(-source._lineWidth / 2),
source._lineLength, source._lineWidth), source._roundness,
                                    source._roundness, QtCore.Qt.RelativeSize)
            painter.restore()

    def start(source):
        source.updatePosition()
        source._isSpinning = True
        source.show()

        if source.parentWidget and source._disableParentWhenSpinning:
            source.parentWidget().setEnabled(False)

        if not source._timer.isActive():
            source._timer.start()
```

```
            source._currentCounter = 0

    def stop(source):
        source._isSpinning = False
        source.hide()

        if source.parentWidget() and source._disableParentWhenSpinning:
            source.parentWidget().setEnabled(True)

        if source._timer.isActive():
            source._timer.stop()
            source._currentCounter = 0

    def setNumberOfLines(source, lines):
        source._numberOfLines = lines
        source._currentCounter = 0
        source.updateTimer()

    def setLineLength(source, length):
        source._lineLength = length
        source.updateSize()

    def setLineWidth(source, width):
        source._lineWidth = width
        source.updateSize()

    def setInnerRadius(source, radius):
        source._innerRadius = radius
        source.updateSize()

    def color(source):
        return source._color

    def roundness(source):
        return source._roundness

    def minimumTrailOpacity(source):
        return source._minimumTrailOpacity

    def trailFadePercentage(source):
        return source._trailFadePercentage

    def revolutionsPersSecond(source):
        return source._revolutionsPerSecond

    def numberOfLines(source):
        return source._numberOfLines

    def lineLength(source):
        return source._lineLength
```

```
    def lineWidth(source):
        return source._lineWidth

    def innerRadius(source):
        return source._innerRadius

    def isSpinning(source):
        return source._isSpinning

    def setRoundness(source, roundness):
        source._roundness = max(0.0, min(100.0, roundness))

    def setColor(source, color=QtCore.Qt.black):
        source._color = QtGui.QColor(color)

    def setRevolutionsPerSecond(source, revolutionsPerSecond):
        source._revolutionsPerSecond = revolutionsPerSecond
        source.updateTimer()

    def setTrailFadePercentage(source, trail):
        source._trailFadePercentage = trail

    def setMinimumTrailOpacity(source, minimumTrailOpacity):
        source._minimumTrailOpacity = minimumTrailOpacity

    def rotate(source):
        source._currentCounter += 1
        if source._currentCounter >= source._numberOfLines:
            source._currentCounter = 0
        source.update()

    def updateSize(source):
        size = (source._innerRadius + source._lineLength) * 2
        source.setFixedSize(size, size)

    def updateTimer(source):
        source._timer.setInterval(int(1000 / (source._numberOfLines *
source._revolutionsPerSecond)))

    def updatePosition(source):
        if source.parentWidget() and source._centerOnParent:
            source.move(int(source.parentWidget().width() / 2 - source.width() /
2),
                        int(source.parentWidget().height() / 2 - source.height() /
2))

    def lineCountDistanceFromPrimary(source, current, primary, totalNrOfLines):
        distance = primary - current
        if distance < 0:
```

```
            distance += totalNrOfLines
        return distance

    def currentLineColor(source, countDistance, totalNrOfLines, trailFadePerc,
minOpacity, colorinput):
        color = QtGui.QColor(colorinput)
        if countDistance == 0:
            return color
        minAlphaF = minOpacity / 100.0
        distanceThreshold = int(math.ceil((totalNrOfLines - 1) * trailFadePerc /
100.0))
        if countDistance > distanceThreshold:
            color.setAlphaF(minAlphaF)
        else:
            alphaDiff = color.alphaF() - minAlphaF
            gradient = alphaDiff / float(distanceThreshold + 1)
            resultAlpha = color.alphaF() - gradient * countDistance
            # обмження альфа в [0,1]
            resultAlpha = min(1.0, max(0.0, resultAlpha))
            color.setAlphaF(resultAlpha)
        return color


class QTableWidget_with_spinner(QtWidgets.QTableWidget):

    def __init__(source, parent=None):
        QtWidgets.QTableWidget.__init__(source, parent)

        source.spinner = QtWaitingSpinner(source)
        source.spinner.setSizePolicy(source.sizePolicy())
```

Файл threads.py

```
from view import QtCore, QtWidgets
import pandas as pd
import seaborn as sns

class Train_Model_WorkerSignals(QtCore.QObject):
    finished = QtCore.pyqtSignal(object, object)
class Train_Model_Thread(QtCore.QRunnable):
    """
    Worker thread

    Inherits from QRunnable to handler worker thread setup, signals and wrap-up.

    """

    def __init__(source, *args, **kwargs):
```

```
        super(Train_Model_Thread, source).__init__()

        source.ml_model = args[0]
        source.model_parameters = args[1]
        source.algorithm_parameters = args[2]
        source.ui = args[3]

        source.args = args
        source.kwargs = kwargs
        source.signals = Train_Model_WorkerSignals()

    @QtCore.pyqtSlot()
    def run(source):
        # тренування моделі
        result = source.ml_model.train(source.model_parameters,
source.algorithm_parameters)
        # виведення результатів
        source.signals.finished.emit(result, source.model_parameters)

class Load_Dataset_WorkerSignals(QtCore.QObject):
    display_message = QtCore.pyqtSignal(object, object, object, object)
    populate_tablewidget_with_dataframe = QtCore.pyqtSignal(object, object)
    update_train_test_shape_label = QtCore.pyqtSignal()
    stop_spinner = QtCore.pyqtSignal(object, object, object)
class Load_Dataset_Thread(QtCore.QRunnable):
    """
    Worker thread

    Inherits from QRunnable to handler worker thread setup, signals and wrap-up.

    """

    def __init__(source, *args, **kwargs):
        super(Load_Dataset_Thread, source).__init__()

        source.ui = args[0]
        source.ml_model = args[1]
        source.file_path = args[2]

        source.args = args
        source.kwargs = kwargs
        source.signals = Load_Dataset_WorkerSignals()

    @QtCore.pyqtSlot()
    def run(source):

        ui = source.ui
        ml_model = source.ml_model

        return_code = source.ml_model.read_dataset(source.file_path)
```

```
        if return_code == 'success':
            source.signals.populate_tablewidget_with_dataframe.emit(source.ui.datas
et_tableWidget, source.ml_model.dataset)
            source.signals.populate_tablewidget_with_dataframe.emit(source.ui.pre_p
rocess_dataset_tableWidget,
                                                                    source.ml_model.d
ataset)
            widgets_to_enable = [ui.plot_radioButton, ui.boxplot_radioButton,
ui.histogram_radioButton,
                                 ui.remove_duplicates_pushButton,
ui.remove_constant_variables_pushButton,
                                 ui.numeric_scaling_pushButton,
ui.remove_outliers_pushButton,
                                 ui.addrule_filter_value_pushButton,
ui.addrule_replace_value_pushButton,
                                 ui.add_input_columns_pushButton,
ui.add_output_columns_pushButton,
                                 ui.remove_preprocessing_rule_pushButton,
                                 ui.clear_preprocessing_rule_pushButton]

            [item.setEnabled(True) for item in widgets_to_enable]

            # оновлення змінних
            if ui.variable_to_plot_comboBox.count() > 0:  # якщо не пуста
                # від'єднання
                ui.variable_to_plot_comboBox.setUpdatesEnabled(False)
                ui.replace_columnSelection_comboBox.setUpdatesEnabled(False)
                ui.filter_columnSelection_comboBox.setUpdatesEnabled(False)
                # очистка
                ui.variable_to_plot_comboBox.clear()
                ui.replace_columnSelection_comboBox.clear()
                ui.filter_columnSelection_comboBox.clear()

                # під'єднання
                ui.variable_to_plot_comboBox.setUpdatesEnabled(True)
                ui.replace_columnSelection_comboBox.setUpdatesEnabled(True)
                ui.filter_columnSelection_comboBox.setUpdatesEnabled(True)

            source.signals.update_train_test_shape_label.emit()

            if ui.available_columns_listWidget.count() != 0:
                ui.available_columns_listWidget.clear()

            if ui.input_columns_listWidget.count() != 0:
                ui.input_columns_listWidget.clear()

            if ui.output_columns_listWidget.count() != 0:
                ui.output_columns_listWidget.clear()
```

```
            ui.preprocess_sequence_listWidget.clear()
            ui.clas_output_colum_comboBox.clear()
            ui.train_model_pushButton.setDisabled(True)

            for each_column in ml_model.dataset.columns:
                ui.variable_to_plot_comboBox.addItem(each_column)  # візуалізація
                ui.replace_columnSelection_comboBox.addItem(each_column)  #
передобробка
                ui.filter_columnSelection_comboBox.addItem(each_column)
                ui.available_columns_listWidget.addItem(each_column)

                if ml_model.column_types_pd_series[each_column].kind in 'iO':
                    ui.clas_output_colum_comboBox.addItem(each_column)
                    ui.clas_output_colum_comboBox.setCurrentIndex(0)

            if len(ml_model.dataset.columns) > 0 :
                ui.variable_to_plot_comboBox.setCurrentIndex(0)
                ui.replace_columnSelection_comboBox.setCurrentIndex(0)
                ui.filter_columnSelection_comboBox.setCurrentIndex(0)

            return

        elif return_code == 'invalidExtension':
            source.signals.display_message.emit(QtWidgets.QMessageBox.Warning,
'Помилка',
                                                'Невірне розширення файлу.
\nСпробуйте .xlsx або .csv файл',
                                                'Помилка')

        elif return_code == 'inputFileException':
            source.signals.display_message.emit(QtWidgets.QMessageBox.Warning,
'Помилка', 'Невірне розширення файлу.', 'Помилка')

        source.signals.stop_spinner.emit(source.ui.dataset_tableWidget,
'stop_spinner' , [])
        source.signals.stop_spinner.emit(source.ui.pre_process_dataset_tableWidget,
'stop_spinner' , [])

class Pre_Process_Dataset_WorkerSignals(QtCore.QObject):
    update_pre_process_tableWidget = QtCore.pyqtSignal(object, object)
    display_message = QtCore.pyqtSignal(object, object, object, object)
class Pre_Process_Dataset_Thread(QtCore.QRunnable):
    """
    Worker thread

    Inherits from QRunnable to handler worker thread setup, signals and wrap-up.

    """

    def __init__(source, *args, **kwargs):
```

```
        super(Pre_Process_Dataset_Thread, source).__init__()

        source.ui = args[0]
        source.ml_model = args[1]

        source.args = args
        source.kwargs = kwargs
        source.signals = Pre_Process_Dataset_WorkerSignals()

    @QtCore.pyqtSlot()
    def run(source):

        ui = source.ui
        ml_model = source.ml_model

        old_pre_processed_dataset = ml_model.pre_processed_dataset.copy()
        ml_model.pre_processed_dataset = ml_model.dataset.copy()

        listwidget = ui.preprocess_sequence_listWidget
        for i in range(listwidget.count()):
            # Отримання даних з віджета списку
            item_data = listwidget.item(i).data(QtCore.Qt.UserRole)

            if item_data['pre_processing_action'] == 'rm_duplicate_rows':
                ml_model.remove_duplicate_rows()
            elif item_data['pre_processing_action'] == 'rm_constant_var':
                ml_model.remove_constant_variables()
            elif item_data['pre_processing_action'] == 'apply_num_scaling':
                ml_model.scale_numeric_values()
            elif item_data['pre_processing_action'] == 'rm_outliers':
                ml_model.remove_outliers(item_data['cut_off'])
            elif item_data['pre_processing_action'] == 'replace_values':
                target_variable = item_data['variable']
                new_value = item_data['new_values']
                old_values = item_data['old_values']
                ml_model.replace_values(target_variable,new_value,old_values)
            elif item_data['pre_processing_action'] == 'apply_filtering':
                filtering_variable = item_data['variable']
                filtering_value = item_data['filtering_value']
                filtering_operator = item_data['filtering_operator']
                ml_model.filter_out_values(filtering_variable,filtering_value,filte
ring_operator)

        table_widget = ui.pre_process_dataset_tableWidget
        filling_dataframe = ml_model.pre_processed_dataset

        if filling_dataframe.empty:
            source.signals.display_message.emit(QtWidgets.QMessageBox.Critical,
'Помилка передобробки',
```

```
                            'Дані правила викреслюють усі елементи бази даних',
'Помилка')
            # Відміна правил
            source.ml_model.pre_processed_dataset = old_pre_processed_dataset
            filling_dataframe = old_pre_processed_dataset
            listwidget.takeItem(listwidget.count() - 1)

        source.signals.update_pre_process_tableWidget.emit(table_widget,filling_dat
aframe)


class Plotting_in_MplWidget_Thread(QtCore.QRunnable):
    """
    Worker thread

    Inherits from QRunnable to handler worker thread setup, signals and wrap-up.

    """

    def __init__(source, *args, **kwargs):
        super(Plotting_in_MplWidget_Thread, source).__init__()

        source.ui = args[0]
        source.target_widget = args[1]
        source.content = args[2]

        source.args = args
        source.kwargs = kwargs

    @QtCore.pyqtSlot()
    def run(source):

        ui = source.ui
        target_widget = source.target_widget
        content = source.content

        target_widget.canvas.axes.clear()
        # Для побудови матриці невідповідностей
        if len(target_widget.canvas.figure.axes) > 1:
            target_widget.canvas.figure.clf()
            target_widget.canvas.axes = target_widget.canvas.figure.add_subplot()
        target_widget.canvas.axes.axis('on')

        if target_widget.objectName() == 'dataVisualisePlot_widget':
            if content['is_categorical']:
                content['data'].value_counts().plot(kind='bar',
ax=target_widget.canvas.axes, grid=False,
                                                    title='Кількість записів')
                target_widget.canvas.axes.tick_params(axis='x', labelrotation=60)
```

```
            elif ui.plot_radioButton.isChecked() and
ui.plot_radioButton.isEnabled():
                content['data'].plot(ax=target_widget.canvas.axes, grid=False,
title='Лінійна діаграма')
            elif ui.boxplot_radioButton.isChecked() and
ui.boxplot_radioButton.isEnabled():
                pd.DataFrame(content['data']).boxplot(ax=target_widget.canvas.axes,
grid=False)
                target_widget.canvas.axes.axes.set_title('Діаграма розмаху')
            elif ui.histogram_radioButton.isChecked() and
ui.histogram_radioButton.isEnabled():
                pd.DataFrame(content['data']).hist(ax=target_widget.canvas.axes,
grid=False)
                target_widget.canvas.axes.axes.set_title('Гістограма')
            else:
                target_widget.canvas.axes.axis('off')

        elif target_widget.objectName() == 'model_train_widget':
            if content['is_regression']:
                if len(content['output_variables']) == 1:
                    pd.DataFrame(content['data']).hist(ax=target_widget.canvas.axes
, grid=False)
                    target_widget.canvas.axes.axes.set_title('Histogram of
Percentage Errors')
                    target_widget.canvas.axes.axes.set_ylabel('Number of
Occurrences')
                    target_widget.canvas.axes.axes.set_xlabel('Percentage Error
(%)')
                else:
                    target_widget.canvas.axes.bar(content['data']['labels'],
content['data']['values'])
                    target_widget.canvas.axes.set_xticklabels(content['data']['labe
ls'], rotation='vertical')
                    target_widget.canvas.axes.axes.set_ylabel('Mean Percentage
Error (%)')
                    target_widget.canvas.axes.axes.set_title('Individual Mean
Percentage Error (%)')
            else:

                is_annot = False if len(content['data']) > 10 else True
                _ = sns.heatmap(content['data'], cmap="YlGnBu", annot=is_annot,
ax=target_widget.canvas.axes)
                target_widget.canvas.axes.tick_params(axis='y', labelrotation=0)
                target_widget.canvas.axes.tick_params(axis='x', labelrotation=60)
                target_widget.canvas.axes.set_xlabel('Actual')
                target_widget.canvas.axes.set_ylabel('Predicted')
                target_widget.canvas.axes.axes.set_title('Матриця
невідповідностей')

        target_widget.canvas.draw()
```