МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Центр післядипломної освіти
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження ефективності застосування дерев рішень для обробки результатів психологічного тестування
(тема)

Виконав:
Студент 2 курсу, групи ІПЗздм-19-1

Майборода В. А.
(прізвище, ініціали)

Спеціальність 121 Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Керівник проф. Смеляков С. В.
(посада, прізвище)

Допускається до захисту

Зав. кафедри _____ Дудар З.В.
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Центр післядипломної освіти
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. Кафедри _____
(підпис)
« 26 » 03 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Майбороди Володимира Андрійовича
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження ефективності застосування дерев рішень для обробки результатів психологічного тестування

затверджена наказом університету від 26.03.2021р. № 34Стз

2. Термін подання роботи до екзаменаційної комісії 11 05 2021р.

3. Вихідні дані до роботи Провести дослідження щодо ефективності застосування дерев рішень для обробки результатів психологічного тестування, розробити та навчити такі дерева для обробки результатів психологічного тестування. Тип психологічного тестування, результат психологічного тестування.

4. Перелік питань, що потрібно опрацювати в роботі мета роботи, постановка задачі, дослідження принципів побудови дерев рішень, побудова моделей, порівняння дерев.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, слайдів, ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)
Титульний лист, Актуальність теми, Постановка задачі, Психологічне

тестування, Набір даних, Підготовка даних, Типи дерев рішень, Етапи побудови дерев рішень, Параметри для пошуку оптимальних дерев рішень, Побудова дерев рішень, Оптимальне класифікаційне дерево рішень, Оптимальне регресійне дерево рішень, Метрики порівняння, Порівняння дерев рішень, Висновки.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Дослідження принципів побудови дерев рішень	проф. Смеляков С. В.		08.04.2021

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Вивчення наявної літератури	03-04-2021	виконав
2	Вивчення технології	08-04-2021	виконав
3	Збір даних	14-04-2021	виконав
4	Проведення експериментів	21-04-2021	виконав
5	Підготовка пояснювальної записки	28-04-2021	виконав
6	Підготовка презентації та доповіді	02-05-2021	виконав
7	Нормоконтроль, рецензування	08-05-2021	виконав
8	Занесення диплому в електронний архів	11-05-2021	виконав
9	Допуск до захисту у зав. кафедри	11-05-2021	виконав

Дата видачі завдання 26 березня 2021р.

Студент _____
(підпис)

Керівник роботи _____ проф. Смеляков В. С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 62 с., 42 рис., 1 табл., 13 джер.

ДЕРЕВА РІШЕНЬ, МАШИННЕ НАВЧАННЯ, КЛАСИФІКАЦІЯ, РЕГРЕСІЯ, ПСИХОЛОГІЧНЕ ТЕСТУВАННЯ, PYTHON, SKLEARN, PANDAS.

Об’єктом дослідження є ефективність дерев рішень для обробки результатів психологічного тестування.

Метою роботи є побудування класифікаційного та регресійного дерев рішень і порівняти їх за швидкістю тренування та точністю передбачень.

Методи розробки базуються на таких технологія, як Python, Sklearn, Pandas.

В результаті роботи було навчено регресійне та класифікаційне дерево рішень на основі результатів психологічного тестування за методикою «Дослідження схильності до ризику» О. Г. Шмельова. Навчені дерева були перевірень на помилки першого та другого роду. Також було зроблено висновки на основі порівнянь класифікаційного та регресійного алгоритмів.

DECISION TREE, MACHINE LEARNING, CLASSIFICATION, REGRASSION, PSYCHOLOGICAL TESTING, PYTHON, SKLEARN, PANDAS.

The object of research is the effectiveness of decision trees for processing the results of psychological testing.

The aim of the work is to construct classification and regression decision trees and compare them in terms of training speed and accuracy of predictions.

Development methods are based on technologies such as Python, Sklearn, Pandas.

As a result, the regression and classification tree of decisions based on the results of psychological testing according to the method of «Study of risk aversion» O. G. Shmelev was taught. Trained trees were tested for errors of the first and second kind. Conclusions were also made based on comparisons of classification and regression algorithms.

Я, Майборода Володимир Андрійович, студент гр. ІПЗздм-19-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Центр післядипломної освіти», заявляю: моя кваліфікаційна робота на тему «Дослідження ефективності дерев рішень для обробки результатів психологічного тестування», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений (а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 8

1. Аналіз проблемної області та постановка задачі 10

1.1 Аналіз проблемної області побудови дерев рішень 10

1.2 Аналіз існуючих аналогів 14

1.3 Постановка задачі 15

2. Дослідження принципів побудови дерев рішень 17

2.1 Структура дерева рішень 18

2.2 Процес побудови 19

2.3 Основні етапи побудови 21

2.3.1 Теоретико-інформаційний критерій 21

2.3.2 Статистичний підхід 21

2.3.3 Середньоквадратична помилка 23

2.3.4 Регресія Пуассона 24

2.3.4 Критерій зупинки алгоритму 24

2.3.5 Відсікання гілок 25

3. Побудова моделей 27

3.1 Збирання даних 27

3.2 Валідація даних 28

3.3 Створення моделей дерев рішень 31

4. Порівняння дерев 43

4.1 Час навчання 43

4.2 Метрики якості 44

Висновок 47

Перелік джерел посилання 48

Додаток А – Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 49

Додаток Б – Звіт результатів перевірки кваліфікаційної роботи на унікальність тексту 50
Додаток В – Експертний висновок нормоконтроль 51
Додаток Г – Слайди презентації 52
Додаток Д – Тези доповіді 60

ВСТУП

Важко уявити сучасний світ без штучного інтелекту. Він зустрічається майже кожного дня, використовується майже в усіх структурах і технологіях, якими ми користуємося: від розпізнання зображень до передбачення тієї чи іншої події. Також ми зустрічаємо штучний інтелект в побуті, наприклад, розумний будинок, в різних смарт-приладах або в іграх, де штучний інтелект допомагає аналізувати кроки гравців, щоб зробити гру цікавішою. Великий вплив штучного інтелекту ми можемо побачити в психології. Часто він допомагає аналізувати і передбачати поведінку людей, їх характер.

Так як штучний інтелект – це великий складний елемент, існують різні підходи, стратегії та технології для його розробки. Велику роль у світі штучного інтелекту відіграють дерева рішень.

Дерева рішень – це не новий підхід. Їх зазвичай використовують в операційних дослідженнях, зокрема в аналізі рішень, щоб допомогти визначити стратегію, яка найімовірніше досягає мети. Також дерева є популярним інструментом машинного навчання, яке широко використовується у створенні штучного інтелекту, як і нейронні мережі. На відміну від нейронних мереж, дерева рішень є зрозумілішими, так як їх можна легко відобразити у вигляді графіку бінарного дерева.

Метою роботи є дослідження ефективності застосування дерев рішень для обробки результатів психологічного тестування. В якості обробки результатів психологічного тестування обрано передбачення схильності студентів до ризику за методикою тестування «Дослідження схильності до ризику», автором якої є О. Г. Шмельов [1] .

В ході атестаційної роботи магістра було:

– проведено опитування студентів за методикою «Дослідження схильності до ризику»;

– сформовано набір даних для навчання та для тестування дерев рішень;

– навчене класифікаційне дерево рішень для передбачення успішності студентів за результатами їх психологічного тестування;

– проведено аналіз класифікаційного дерева рішень на помилки першого та другого роду;

– навчене регресійне дерево рішень для передбачення успішності студентів за результатами їх психологічного тестування;

– проведено зрівняння дерев за швидкістю навчання та за відсотком помилок у передбаченні.

За результатами атестаційної роботи магістра було розроблено презентацію Також за результатами роботи було створено та опубліковано тези на 25-ому Міжнародному молодіжному форумі «Радіоелектроніка і молодь у XXI столітті».

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ

1.1 Аналіз проблемної області побудови дерев рішень

Важко уявити сучасний світ без машинного навчання та штучного інтелекту. Він набув великої популярності і використовується майже у всіх шарах нашого життя. Ми зустрічаємо його кожен день від домашніх приладів до фінансових систем. Навіть коли просто заходимо в інтернет подивитися новини, вбудовані алгоритми штучного інтелекту роблять аналіз та пропонуються нам новини, які були б вам найцікавішими.

Інтегрування штучного інтелекту в програму або гру не є тривіальною задачею. Для того, щоб це зробити, перш за все необхідно зрозуміти, яку саме бізнес задачу буде вирішувати алгоритм. Потім ми повинні отримати дані, на основі яких робити аналіз. Це одна з ключових задач в побудові штучного інтелекту, адже від даних залежить якість та точність вирішення тієї чи іншої задачі. Зазвичай задачею створення набору даних займаються вчені даних. На перший погляд, задача збирання та генерування даних є простою, але сучасні системи складаються з десятків і більше сервісів і відповідних сховищ з даними. Вчені даних мають зібрати дані, щоб на основі них можна було вирішити конкретну проблему.

Наступним кроком йде статистичний аналіз набору даних. Тобто статисти роблять аналіз даних, намагаються знайти закономірності та оцінити цілісність даних. Перш за все необхідно побудувати математичну модель і тільки після цього йде вибір підходу, на основі якого буде вирішено цю задачу. Це може бути як і лінійна регресія, так і нейронні мережі або дерева рішень.

Щоб вибрати той чи інший підхід, потрібно провести аналіз або порівняння за різними критеріями, адже кожен підхід має свої переваги і недоліки.

Лінійна регресія – один з найпростіших інструментів статистичного моделювання, але саме в його простоті і полягає його ефективність [2]. Для початку необхідно зрозуміти значення терміну. У спрощеному вигляді регресійний аналіз

можна визначити як спосіб опису взаємовідносин між залежною змінною і набором незалежних змінних. Наприклад, за допомогою цього підходу при наявності набору даних про ріст, вагу, стать і інших фізичних параметрів дитини ми можемо описати вплив цих змінних на цільову змінну – як ця дитина буде виглядати в дорослому віці.

Регресія моделює числові відносини, тобто вихідні дані регресійного аналізу є завжди числовими коефіцієнтами. Цим вона відрізняється від категоризації, результатом якої є клас об'єктів або мітка для вихідних даних. Друга частина терміну «лінійна регресія» підкреслює лінійний характер залежності між змінними.

Моделі лінійної регресії дуже популярні в різних сферах досліджень завдяки швидкості їх створення і простоті інтерпретації. Завдяки можливості перетворення даних, вони можуть бути використані для моделювання широкого спектру залежностей. Завдяки простоті форми, в порівнянні з нейронними мережами, їх статистичні параметри легко піддаються аналізу і порівнянню, що дозволяє витягувати з них цінну інформацію. Лінійна регресія використовується не тільки в прогностичних цілях; вона також показала свою ефективність в описі систем.

Головний недолік лінійної регресії полягає в тому, що вона може моделювати тільки прямі лінійні залежності, в той час як часто виникає необхідність створення моделей інших типів відносин між даними.

Нейронна мережа – це послідовність нейронів, з'єднаних між собою синапсами [3]. Структура нейронної мережі прийшла в світ програмування прямо з біології. Завдяки такій структурі, машина отримує можливість аналізувати і навіть запам'ятовувати різну інформацію. Нейронні мережі також здатні не тільки аналізувати вхідну інформацію, а й відтворювати її зі своєї пам'яті [4].

Нейронні мережі використовуються для вирішення складних завдань, які вимагають аналітичних обчислень подібних тим, що робить людський мозок. Найпоширенішими застосуваннями нейронних мереж є:

– класифікація – розподіл даних по параметрах. Наприклад, на вхід дається набір людей і потрібно вирішити, кому з них давати кредит, а кому ні. Цю роботу

може зробити нейронна мережа, аналізуючи таку інформацію, як: вік, платоспроможність, кредитна історія і т. д.;

– передбачення – можливість прогнозувати наступний крок. Наприклад, зростання або падіння акцій, ґрунтуючись на ситуації на фондовому ринку;

– розпізнавання – в даний час широко застосовується в нейронних мережах [5]. Використовується в Google, коли йде пошук фото, або в камерах телефонів, коли воно визначає положення вашого обличчя і виділяє його.

Крім можливості вирішувати новий клас задач, нейромережі мають ряд значних переваг. Зрозуміти, звідки вони беруться, дуже просто. Всі плюси нейронних мереж є наслідками плюсів біологічних нейронних мереж, так як саму модель обробки інформації практично не було змінено.

Нейронні мережі використовується у вирішенні багатьох задач. Вони набули великої популярності за рахунок великої кількості переваг. Нейронні мережі здатні коректно функціонувати, навіть якщо на вході дані являються зашумленими. Зашумлені дані – це дані, які пошкоджені, спотворені або мають низьке відношення сигнал/шум. Неправильні процедури (або неправильно задокументовані процедури) для вирахування шуму в даних можуть привести до помилок в точності або хибним висновкам. Також нейронні мережі можуть підлаштовуватися під навколишнє оточення, яке змінюється. Нейронні мережі здатні нормально функціонувати навіть при досить серйозних пошкодженнях. Нейронні мережі вирішують завдання швидше інших алгоритмів.

У нейронних мереж є ряд серйозних недоліків, які теж можна вивести з біологічних нейронних мереж. Варто зауважити, що нейронні мережі, незважаючи на широкий спектр завдань, які вони можуть вирішувати, все ж залишаються лише корисним додатковим функціоналом. На першому місці завжди стоять комп'ютерні програми.

Нейронні мережі не здатні давати точні і однозначні відповіді. Ви ніколи не будете отримувати точні відповіді. Гарна новина полягає в тому, що рідко зустрічаються завдання, в яких потрібно застосовувати нейронні мережі і одночасно отримувати точні відповіді. Нейронні мережі не можуть вирішувати

завдання по кроках. Нейронні мережі не здатні вирішувати обчислювальні завдання.

Дерева рішень є одним із найбільш ефективних інструментів інтелектуального аналізу даних, який дозволяє вирішувати завдання класифікації і регресії [6].

Дерево рішень – логічна схема, що надає можливість отримати остаточне рішення про класифікацію об'єкта після відповідей на ієрархічно організовану систему питань. Вони являють собою ієрархічні деревоподібні структури, що складаються з вирішальних правил виду «Якщо ..., то ...». Правила автоматично генеруються в процесі навчання на навчальній множині і, оскільки вони формулюються практично на природній мові, дерева рішень як аналітичні моделі є більш вербалізованими і інтерпретованими, ніж, скажімо, нейронні мережі.

Власне, саме дерево рішень – це спосіб подачі вирішальних правил в ієрархічній структурі, що складається з елементів двох типів – вузлів і листя. У вузлах знаходяться вирішальні правила і проводиться перевірка відповідності прикладів цього правила з якого-небудь атрибуту навчальної множини.

Основна сфера застосування дерев рішень – підтримка процесів прийняття управлінських рішень, використовувана в статистиці, аналізі даних і машинному навчанні. Завданнями, які розв'язуються за допомогою даного апарату, є:

– класифікація – віднесення об'єктів до одного з заздалегідь відомих класів. Цільова змінна повинна мати дискретні значення;

– регресія (чисельне пророкування) – прогноз числового значення незалежної змінної для заданого вхідного вектора;

– опис об'єктів – набір правил в дереві рішень, який дозволяє компактно описувати об'єкти. Тому замість складних структур, що описують об'єкти, можна зберігати дерева рішень.

Головним недоліком дерев рішень є те, що вони чутливі до шумів у вхідних даних. Невеликі зміни навчальної вибірки можуть привести до глобальних корегувань моделі, що позначиться на зміні правил класифікації і інтерпретованої моделі. Також можливе перенавчання дерева рішень, через що, доводиться

вдаватися до методу «відсікання гілок», установці мінімального числа елементів в листі дерева або максимальної глибини дерева. Не потрібно ігнорувати і той факт, що пошук оптимального дерева рішень є доволі складною процедурою, яка призводить до необхідності використання евристик, які в кінцевому підсумку не дають 100-відсоткової гарантії знаходження оптимального дерева.

Розглянувши основні проблеми, що виникають при побудові дерев, було б несправедливо не згадати про їх переваги. І однією з головних переваг є те, що дерева рішень формують чіткі і зрозумілі правила класифікації. Тобто вони виносять рішення на основі простих умов, які легко зрозуміти людині. Також вони здатні генерувати правила в областях, де фахівцеві важко формалізувати свої знання. Дерева можуть легко візуалізуватися, тобто можуть «інтерпретуватися» не тільки як модель в цілому, але і як прогноз для окремого тестового суб'єкта. Не потрібно ігнорувати той факт, що дерева швидко навчаються і прогнозують, на відміну від інших популярних алгоритмів. Побудувати дерево рішень простіше за рахунок того, що не потрібно багато параметрів моделі. Також вони підтримують як числові, так і категоріальні ознаки.

Як ми бачимо, існує велика кількість алгоритмів і підходів для вирішення задач машинного навчання і не менш важливим є вибір того підходу, який краще вирішить конкретну задачу за параметрами, такими як швидкість навчання, точність тощо. Часто аналітики будують рішення задач, використовуючи різні підходи для того, щоб вибрати найкращий із можливих.

1.2 Аналіз існуючих аналогів

Порівняння різних алгоритмів за критеріями якості не новий виклик. Науковці у своїх дослідах проводять різні експерименти перевіряючи, який з підходів краще вирішить конкретну задачу. Цим самим вони допомагають іншим

розробникам тим, що вони можуть посилатися на вже готові результати і зменшити час на аналіз того чи іншого підходу.

Часто великі технологічні корпорації, такі як Google, Amazon, Microsoft, роблять свої власні досліди або фінансують університети, щоб ті проводили дослідження в цій області.

Вже існує велика кількість наукових публікацій, які порівнюють як різні архітектури одного підходу, так і різні алгоритми між собою. Так часто порівнюють нейронні мережі, дерева рішень та лінійну регресію між собою для вирішення різних задачах [7]. Наприклад, зрозуміти, який алгоритм швидше аналізуватиме зображення або що краще використовувати для передбачення цін на продукти чи паливо тощо.

Була проведена велика кількість наукових досліджень, які порівнюють різні підходи і архітектури для побудови нейронних мереж. Одним із таких є порівняння сучасних архітектур глибоких нейронних мереж для прогнозування цін на енергію. Де Франческо Кордоні розробив чотири нейронні мережі на основі чотирьох підходів: багатошаровий персептрон (MLP), згорткова нейронна мережа (CNN), довго-короткий час пам'яті (LSTM) та мережа з накопиченням (CNN-LSTM) [8]. Метою цього дослідження було порівняти ці 4 підходи за продуктивністю.

Оскільки дерева рішень – це популярний та потужний алгоритм, який має велику популярність, існує велика кількість наукових статей та публікацій з порівнянням різних підходів для конкретних задач, наприклад, стаття професорів з Масачусецького медичного університету про порівняння класифікаційних та регресійних дерев рішень у вирішенні проблем публічної медицини [9].

1.3 Постановка задачі

Метою роботи є дослідження ефективності застосування дерев рішень для обробки результатів психологічного тестування.

Практичною задачею роботи є побудова класифікаційного та регресійного дерев рішень, які будуть передбачати успішність студентів на основі їх психологічного тестування.

За основу психологічного тестування буде взято опитування двохсот студентів використовуючи методику «Дослідження схильності до ризику».

Для досягнення поставленої мети необхідно:

– провести опитування студентів за методикою «Дослідження схильності до ризику»;

– обробити отримані результати та конвертувати їх в набір даних;

– сформувати набір даних для навчання та для перевірки дерев рішень;

– навчити класифікаційне дерево рішень для передбачення успішності студентів за результатами їх психологічного тестування;

– провести аналіз класифікаційного дерева рішень на помилки першого та другого роду;

– навчити регресійне дерево рішень для передбачення успішності студентів за результатами їх психологічного тестування;

– провести порівняння дерев за швидкістю навчання та за відсотком помилок у передбаченні.

2 ДОСЛІДЖЕННЯ ПРИНЦИПІВ ПОБУДОВИ ДЕРЕВ РІШЕНЬ

Дерева рішень є одним з найбільш ефективних інструментів інтелектуального аналізу даних і самий корінь аналітики, які дозволяють вирішувати завдання класифікації і регресії.

Вони являють собою ієрархічні деревоподібні структури, що складаються з вирішальних правил виду «Якщо ..., то ...». Правила автоматично генеруються в процесі навчання на навчальній множині і, оскільки вони формуються практично на природній мові (наприклад, «Якщо обсяг продажів більше 1000 шт., То товар перспективний»), дерева рішень як аналітичні моделі більш верболізованих і інтерпретованих, ніж, скажімо, нейронні мережі.

Оскільки правила в деревах рішень виходять шляхом узагальнення безлічі окремих спостережень (навчальних прикладів), що описують предметну область, то за аналогією з відповідним методом логічного висновку їх називають індуктивними правилами, а сам процес навчання індукцією дерев рішень.

У навчальній множині для прикладів має бути задано цільове значення, тому що дерева рішень є моделями, що будуються на основі навчання з учителем. При цьому, якщо цільова змінна дискретна (мітка класу), то модель називають деревом класифікації, а якщо безперервна, то деревом регресії.

Основні ідеї, що послужили поштовхом до появи і розвитку дерев рішень, були закладені в 1950-х роках в області досліджень моделювання людської поведінки за допомогою комп'ютерних систем. Серед них слід виділити такі роботи: К. Ховеленд «Комп'ютерне моделювання мислення» [10], Е. Ханта «Експерименти по індукції» [11].

Подальший розвиток дерев рішень як моделей, які можуть самостійно навчатися, для аналізу даних пов'язано з іменами Джона Р. Квінлена [12], який розробив алгоритм ID3 і його вдосконалені модифікації C4.5 і C5.0, а також Лео Бреймана [13], який запропонував алгоритм CART і метод випадкового лісу.

2.1 Структура дерева рішень

Власне, саме дерево рішень – це спосіб подачі вирішальних правил в ієрархічній структурі, що складається з елементів двох типів – вузлів і листя. У вузлах знаходяться вирішальні правила і проводиться перевірка відповідності прикладів цього правила з якого-небудь атрибуту навчальної множини.

У найпростішому випадку, в результаті перевірки, безліч прикладів, які потрапили в вузол, розбивається на дві підмножини, в одну з яких потрапляють приклади, що задовольняють правилу, а в іншу – що не задовольняють. Приклад дерева рішень, яке вирішує, чи за страхувати автомобіль, чи ні, зображено на рисунку 2.1.

Рисунок 2.1 – Приклад дерева рішень

Потім до кожної підмножини знову застосовується правило і процедура рекурсивно повторюється, поки не буде досягнуто деякої умови зупинки алгоритму. В результаті в останньому вузлі перевірка і розбиття не проводиться, і

він оголошується листком. Лист визначає рішення для кожного прикладу, який в нього потрапив. Для дерева класифікації це клас, що асоціюється з вузлом, а для дерева регресії – відповідний листу модальний інтервал цільової змінної.

Таким чином, на відміну від вузла, в листі міститься не правило, а підмножина об'єктів, які відповідають всім правилам гілки, яка закінчується даними листом.

Очевидно, щоб потрапити в лист, приклад повинен відповідати всім правилам, які лежать на шляху до цього листа. Оскільки шлях в дереві до кожного листу єдиний, то й кожен приклад може потрапити тільки в один лист, що забезпечує єдність розв'язку.

2.2 Процес побудови

Процес побудови дерев рішень полягає в послідовному, рекурсивному розбитті навчальної множини на підмножини з застосуванням вирішальних правил в вузлах. Процес розбиття триває до тих пір, поки всі вузли в кінці всіх гілок не будуть оголошені листям. Оголошення вузла листом може статися природним чином (коли він буде містити єдиний об'єкт або об'єкти тільки одного класу) або після досягнення деякої умови зупинки, що задається користувачем (наприклад, мінімально допустиму кількість прикладів в вузлі або максимальна глибина дерева).

Алгоритми побудови дерев рішень відносять до категорії так званих жадібних алгоритмів. Жадібними називаються алгоритми, які допускають, що локально-оптимальні рішення на кожному кроці (розбиття в вузлах), призводять до оптимального підсумкового рішення. У разі дерев рішень це означає, що якщо один раз був обраний атрибут і по ньому було вироблено розбиття на підмножини, то алгоритм не може повернутися назад і вибрати інший атрибут, який дав би краще

підсумкове розбиття. Тому на етапі побудови можна сказати, що обраний атрибут в кінцевому підсумку забезпечить оптимальне розбиття.

В основі більшості популярних алгоритмів навчання дерев рішень лежить принцип «розділяй і володарюй». Алгоритмічно цей принцип реалізується в такий спосіб. Нехай задано навчальну множину S, що містить n прикладів, для кожного з яких задана мітка класу $C_i(i = 1..n)$, і m атрибутів $A_i(i = 1..m)$ які, як передбачається, визначають приналежність об'єкта до того чи іншого класу. Тоді можливі три випадки:

– всі приклади множини S мають однакову мітку класів C_i (тобто всі навчальні приклади відносяться тільки до одного класу). Очевидно, що навчання в цьому випадку не має сенсу, оскільки всі приклади, що пред'являються моделі, будуть одного класу, який і «навчиться» розпізнавати модель. Саме дерево рішень в цьому випадку буде являти собою лист, асоційований з класом C_i. Практичне використання такого дерева безглуздо, оскільки будь-який новий об'єкт воно буде відносити тільки до цього класу;

– множина S взагалі не містить прикладів, тобто є порожньою безліччю. В цьому випадку для нього теж буде створений лист (застосовувати правило, щоб створити вузол, до порожньої безлічі безглуздо), клас якого буде обраний з іншої безлічі (наприклад, клас, який найбільш часто зустрічається в батьківській безлічі);

– множина S містить навчальні приклади всіх класів C_i. В такому випадку потрібно розбити множину S на підмножини, асоційовані з класами. Для цього вибирається один з атрибутів A_j множини S, який містить два і більше унікальних значення (a_1, a_2, …, a_p), де p – число унікальних значень ознаки. Потім множина S розбивається на p підмножин (S_1, S_2, …, S_p), кожна з яких включає приклади, що містять відповідне значення атрибута. Потім вибирається наступний атрибут і розбиття повторюється. Ця процедура буде рекурсивно повторюватися до тих пір, поки всі приклади в результуючих підмножинах не опиняться одного класу.

Описана вище процедура лежить в основі багатьох сучасних алгоритмів побудови дерев рішень. Очевидно, що при використанні даної методики, побудова дерева рішень відбуватиметься зверху вниз (від кореневого вузла до листя).

2.3 Основні етапи побудови

При формуванні правила для розбиття в черговому вузлі дерева необхідно вибрати атрибут, за яким це буде зроблено. Загальне правило для цього можна сформулювати наступним чином: обраний атрибут повинен розбити безліч спостережень в вузлі так, щоб результуючі підмножини містили приклади з однаковими мітками класу або були максимально наближені до цього, тобто кількість об'єктів з інших класів («домішок») в кожному з цих множин було якомога менше. Для цього були обрані різні критерії, найбільш популярними з яких стали теоретико-інформаційний та статистичний.

2.3.1 Теоретико-інформаційний критерій

Теоретико-інформаційний критерій, як випливає з назви, – критерій, заснований на поняттях теорії інформації, а саме інформаційної ентропії, яка вираховується за формулою (2.1).

$$H = -\sum_{i=1}^{n} \frac{N_i}{N} \log\left(\frac{N_i}{N}\right), \tag{2.1}$$

де n – число класів у вихідній підмножині, N_i – число прикладів і-го класу, N – загальна кількість прикладів у підмножині.

Таким чином, ентропія може розглядатися як міра неоднорідності підмножини за представленими в ньому класів. Коли класи представлені в рівних частках і невизначеність класифікації найбільша, ентропія також максимальна. Якщо все приклади в вузлі відносяться до одного класу, тобто $N_i = N$, логарифм від одиниці звертає ентропію в нуль.

Таким чином, найкращим атрибутом розбиття A_j буде той, який забезпечить максимальне зниження ентропії результуючого підмножини щодо батьківського. На практиці, однак, говорять не про ентропію, а про величину, зворотню їй, яка називається інформацією. Тоді найкращим атрибутом розбиття буде той, який забезпечить максимальний приріст інформації результуючого вузла щодо вихідного, що зображено на формулі (2.2).

$$Gain(A) = Info(S) - Info(S_A), \tag{2.2}$$

де $Info(S)$ – інформація, пов’язана з підмножиною S до розбиття, $Info(S_A)$ – інформація, пов'язана з підмножиною, отриманими при розбитті по атрибуту A.

Таким чином, завдання вибору атрибута розбиття в вузлі полягає в максимізації величини $Gain(A)$, званої приростом інформації (від англ. gain - приріст, збільшення). Тому сам теоретико-інформаційний підхід відомий як критерій приросту інформації.

2.3.2 Статистичний підхід

В основі статистичного підходу лежить використання індексу Джині Статистичний сенс даного показника в тому, що він показує, наскільки часто випадково обраний приклад навчальної вибірки буде розпізнано неправильно, що цільові значення в цій множині були взяті з певного статистичного розподілу. Джині є мірою того, як часто випадково вибраний елемент із набору буде неправильно позначений, якщо він буде випадково позначений відповідно до розподілу міток у підмножині.

Таким чином, індекс Джині фактично показує відстань між двома розподілами – розподілом цільових значень і розподілом передбачень моделі.

Очевидно, що чим менше дана відстань, тим краще буде працювати майбутня модель. Індекс Джині може бути розрахований за формулою (2.3).

$$Gini(Q) = 1 - \sum_{i=1}^{n} p_i^2 , \tag{2.3}$$

де Q – результуюча множина, n – кількість класів в ньому, p_i – ймовірність і-го класу (виражена як відносна частота прикладів відповідного класу). Очевидно, що даний показник змінюється від 0 до 1. При цьому він дорівнює 0, якщо всі приклади Q відносяться до одного класу, і дорівнює 1, коли класи представлені в рівних пропорціях і різновірогідні.

2.3.3 Середньоквадратична помилка

Середньоквадратична помилка (MSE) – це один із найпростіших і найпопулярніших способів вимірювання якості роботи моделі. Маючи правильну відповідь y і передбачення $\hat{y}$, можна обчислити їх різницю. Якщо скласти такі різниці для всіх відповідей, зведені в квадрат, і розділити на кількість елементів вибірки, вийде число, що характеризує якість моделі. Цю метрику можна вирахувати за формулою (2.4).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.4}$$

В ідеальному випадку, коли всі відповіді передбачені точно, MSE буде дорівнювати нулю.

Але важко зрозуміти, що показує число, наприклад, 69.2. На жаль, однозначно відповісти на ці питання неможливо.

Для того, щоб зрозуміти, наскільки добре працює модель, потрібно провести серію прогнозів із різними параметрами, порівняти середньоквадратичну помилку результатів між собою і вибрати найкращий варіант.

Середня квадратична помилка - це метрика оцінки моделі, яка часто використовується з регресійними моделями. Середня квадратична помилка моделі щодо тестового набору є середнім значенням квадратичних помилок прогнозування для всіх екземплярів тестового набору. Помилка передбачення - це різниця між справжнім значенням та передбачуваним значенням для екземпляра.

Коли працюєш з MSE, потрібно завжди пам'ятати про те, що ця оцінка абсолютно нестійка до викидів.

2.3.4 Регресія Пуассона

Регресія Пуассона – це узагальнена лінійна модель форми регресійного аналізу, що використовується для моделювання даних про підрахунок та таблиць непередбачених ситуацій. Регресія Пуассона передбачає, що змінна відповіді Y має розподіл Пуассона, і передбачає, що логарифм її очікуваного значення може бути змодельований лінійною комбінацією невідомих параметрів. Модель регресії Пуассона іноді називають лінійно-часовою моделлю, особливо коли вона використовується для моделювання таблиць непередбачених ситуацій. Регресія Пуассона розраховується за формулою (2.5).

$$\ln(\lambda) = \beta_0 + \sum_p \beta_j x_j, \tag{2.5}$$

де $x_1,\ldots,x_p$ – набір з p незалежних змінних, β_0 – математичне очікування Y при рівності нулю всіх предикатів x_j, β_j – коефіцієнт незалежних змінних.

2.3.4 Критерій зупинки алгоритму

Теоретично, алгоритм навчання дерева рішень буде працювати до тих пір, поки в результаті не будуть отримані абсолютно «чисті» підмножини, в кожній з яких будуть приклади одного класу. Правда, можливо, при цьому буде побудовано дерево, в якому для кожного прикладу буде створено окремий лист. Очевидно, що таке дерево виявиться марним, оскільки воно буде перенавчано – кожному, наприклад, буде відповідати свій унікальний шлях в дереві, а отже і набір правил актуальний тільки для даного прикладу.

Перенавчання в разі дерева рішень веде до тих самих наслідків, що і для нейронної мережі – точне розпізнавання прикладів, які беруть участь в навчанні і повна неспроможність на нових даних. Крім цього, перенавчені дерева мають дуже складну структуру, і тому їх складно інтерпретувати.

Очевидним рішенням проблеми є примусова зупинка побудови дерева, поки воно не стало перенавченим. Для цього розроблені такі підходи:

– рання зупинка – алгоритм буде зупинений, як тільки буде досягнуто задане значення деякого критерію, наприклад, процентної частки правильно розпізнаних прикладів. Єдиною перевагою підходу є зниження часу навчання. Головним недоліком є те, що рання зупинка завжди робиться на шкоду точності дерева, тому багато авторів рекомендують віддавати перевагу відсікання гілок;

– обмеження глибини дерева – завдання максимального числа розбиття в гілках, після досягнення якого навчання зупиняється. Даний метод також веде до зниження точності дерева;

– завдання мінімально допустимого число прикладів у вузлі – заборонити алгоритму створювати вузли з числом прикладів менше заданого (наприклад, 5). Це дозволить уникнути створення тривіальних розбиття і, відповідно, малозначущих правил.

Усі перераховані підходи є евристичними, тобто не гарантують кращого результату або взагалі працюють тільки в якихось окремих випадках. Тому до їх використання слід підходити з обережністю. Будь-яких обґрунтованих

рекомендацій по тому, який метод краще працює, в даний час теж не існує. Тому аналітикам доводиться використовувати метод проб і помилок.

2.3.5 Відсікання гілок

Як було зазначено вище, якщо «зростання» дерева обмежити, то в результаті буде побудовано складне дерево з великим числом вузлів і листя. Як наслідок, воно буде важко інтерпретуватися. У той же час, вирішальні правила в таких деревах, що створюють вузли, в які потрапляють два-три приклади, виявляються малозначущими з практичної точки зору.

Набагато краще мати дерево, що складається з малої кількості вузлів, яким би відповідало велику кількість прикладів з навчальної вибірки. Тому становить інтерес підхід, альтернативний ранньої зупинки – побудувати всі можливі дерева і вибрати те з них, яке при розумній глибині забезпечує прийнятний рівень помилки розпізнавання, тобто знайти найбільш вигідний баланс між складністю і точністю дерева.

Альтернативним підходом є так зване відсікання гілок або обрізання. Обрізання – це метод стиснення даних в алгоритмах машинного навчання і пошуку, який зменшує розмір дерев рішень за рахунок видалення некритичних і надлишкових ділянок дерева для класифікації примірників. Обрізка знижує складність остаточного класифікатора і, отже, підвищує точність прогнозів за рахунок зменшення перенавчання. Обрізання гілок, очевидно, проводиться в напрямку, протилежному напрямку росту дерева, тобто від низу до верху, шляхом послідовного перетворення вузлів у листя. Перевагою відсікання гілок в порівнянні з ранньої зупинкою є можливість пошуку оптимального співвідношення між точністю і зрозумілістю дерева. Недоліком є більший час навчання через необхідність спочатку побудувати повне дерево.

3 ПОБУДОВА МОДЕЛЕЙ

3.1 Збирання даних

За основу даних обрано результати тестування за методикою «Дослідження схильності до ризику», а також оцінки студентів в університеті. Згідно з методикою, кожен студент проходить тест, який складається із п’ятдесяти запитань. Наявні тільки відповіді «так» або «ні». У тесті містяться десять перевірочних запитань, за якими визначається, чи студент відповідав чесно або брехав, та сорок запитань самого тестування. У результаті ми отримуємо два значення, де перше відповідає за достовірність опитування, та друге відповідає результатам психологічного тестування.

Спершу було зроблено підрахунок шкали достовірності. Перевірочні запитання йдуть під такими номерами: 1, 5, 10, 16, 20, 24, 29, 35, 36, 46. Якщо відповіді на ці питання «ні», то вони сумуються. Якщо кількість відповідей «ні» більше восьми, то тест рахується не правдивим, і в колонку з результатами передаємо кількість зі знаком «-» для спрощення валідації набору даних. Якщо кількість перевірочних запитань менше восьми, то підраховуються результати основного тесту. Відповідно перевірочним даним, було інкрементовано результат тесту, якщо відповідь студента відповідає так само, як значення в даних тесту. Дані тесту наступні: 2 – «так», 3 – «так», 4 – «так», 6 – «ні», 7 – «ні», 8 – «ні», 9 – «так», 11 – «так», 12 – «так», 13 – «ні», 14 – «так», 15– «ні», 17 – «так», 18 – «так», 19 – «ні», 21 – «ні», 22 – «ні», 23 – «так», 25 – «так», 26 – «так», 27 – «так», 28 – «ні», 30 – «ні», 31 – «так», 32 – «ні», 33 – «так», 34 – «так», 37 – «так», 38 – «так», 39 – «так», 40 – «так», 41 – «так», 42 – «ні», 43 – «ні», 44 – «так», 45 – «ні», 47 – «ні», 48 – «так», 49 – «ні», 50 – «так». Підраховані результати було занесено в Excel таблицю. Далі було отримано оцінки студентів з їхніх предметів, середній бал і кількість боргів за перші два курси навчання в університеті.

У підсумку було опитано двісті студентів. Результати психологічного тестування та оцінки студентів були збережені в таблицю Excel. Приклад отриманих даних зображено на рисунку 3.1.

Результати тесту	ВМ 1 семетр	К Д М	О П	ІМ 1 семетр	Б Ж
11	93	92	91	77	88
24	68	82	81	94	94
6	60	60	65	90	93
15	84	72	90	77	65
-8	75	75	63	86	82
-9	66	76	77	84	85
15	90	85	83	92	99
17	90	90	100	91	100
17	90	75	79	72	78

Рисунок 3.1 – Приклад створеного набору даних

Наступним кроком дані було конвертовано в формат CSV для подальшої обробки їх програмним кодом.

3.2 Валідація даних

Для валідації та подальшої обробки даних було обрано мову програмування Python версії 3.8, так як вона є найпопулярнішою мовою програмування для машинного навчання на даний час. Python – високорівненва мова програмування загального призначення з динамічної строгою типізацією і автоматичним управлінням пам'яттю, орієнтована на підвищення продуктивності розробника, читання коду і його якості, а також на забезпечення переносимості написаних на ній програм. Мова є повністю об'єктно-орієнтованою – усе є об'єктами. Незвичайною особливістю мови є виділення блоків коду пробільними відступами. Синтаксис ядра мови мінімалістичний, за рахунок чого на практиці рідко виникає

необхідність звертатися до документації, сама ж мова відома як інтерпретована і використовується в тому числі для написання скриптів. Недоліками мови є більш низька швидкість роботи і більш високе споживання пам'яті написаних на ній програм в порівнянні з аналогічним кодом, написаним на компільованих мовах, таких як C або C++.

Також для простішої роботи з даними обрано бібліотеку Pandas для Python.

Для валідаці дані у форматі CSV було завантажено в пам’ять, для цього було використано метод read_csv. Приклад коду наведено на рисунку 3.2.

```
# вичитуємо дані
source_data = pd.read_csv("./source_data.csv")
```

Рисунок 3.2 – Вичитування даних в пам’ять

Дані з CSV файлу було збережено у змінну source_data. Бібліотека Pandas автоматично зберегла дані у форматі DataFrame для подальшої їхньої обробки.

Під час валідації нам було видалено недостовірні дані, тобто результати тестів, які не пройшли достовірність. Результати тесту вважаються недостовірними, якщо в колонці «Результати тесту» знаходиться значення менше мінус восьми. На рисунку 3.3 наведено код, який видаляє недостовірні дані.

```
# видалення недостовірних даних
source_data = source_data[source_data['Результати тесту'] <= -8]
```

Рисунок 3.3 – Видалення недостовірних даних

Після видалення недостовірних даних у змінній source_data знаходяться тільки достовірні результати тесту, на яких можна будувати дерева рішень. Наступним кроком було розглянуто атрибути, за якими відбуватиметься навчання та перевірка. Оскільки в предметах, які знаходяться у наборі даних, є предмети, які студенти обирали персонально, то в інших студентів оцінки по деяким з них відсутні. Оскільки дерева рішень на пропущених даних навчаються погано і

матимуть великий відсоток помилок, їх було опрацьовано. Так як для обробки даних є два можливих варіанти обробки, де перший – це задавання значення за замовчуванням, що у даному випадку не відповідає умові завдання, оскільки для цього потрібно передбачати, яку оцінку міг би отримати студент по тому чи іншому предмету, тому залишається тільки другий варіант – це видалення предметів, за якими студент не має оцінки.

Перш за все, нам було вирахувано атрибути, в яких є пропуски в даних. Для цього використовувалася бібліотека Pandas, а саме метод isnull. Приклад коду для підрахунку колонок, де пропущені дані, наведено на рисунку 3.4.

```
# вираховуємо колонки де пропущені данні
null_sum = source_data.isnull().sum()
```

Рисунок 3.4 – Приклад коду вирахування колонок з пропущеними даними

Вирахувані результати було збережено у змінну null_sum для подальшого використання. У змінній дані зберігаються у форматі об’єкту Series, який використовує бібліотека Pandas. Приклад з генерованими даними продемонстровано на рисунку 3.5.

```
Л М В                    0
Г д и з                115
К р П                  102
О І Г                  152
П Е С Е А              141
П С П І                131
П У п р                 95
```

Рисунок 3.5 – Колонки з кількістю пропущених даних в них

Наступним кроком було видалено колонки з пропущеними даними. Для цього було згенеровано масив з назвами колонок, в яких кількість пропущених значень більше нуля. Результат попередньої операції було збережено у змінну

bad_columns. Потім колонки було видалено з об'єкту source_data за допомогою метода drop. Приклад цієї логіки наведено на рисунку 3.6.

```
# генеруємо масив з поганих колонок
bad_columns = np.array(null_sum.where(null_sum > 0).dropna().axes)[0]

# видаляємо невалідні дані
validated_data = source_data.drop(bad_columns, axis=1)
```

Рисунок 3.6 – Видалення невалідних колонок

Результати попередніх дій збережено у змінну validated_data. Для подальшого використання цих даних їх було збережено у файл з форматом CSV (див. рис. 3.7).

```
# збереження від валідованих даних у CSV файл
validated_data.to_csv("./validated_data.csv")
```

Рисунок 3.7 – Збереження даних у CSV файл

У підсумку було відфільтровано та відвалідовано вихідний набір даних та збережено у файл формату CSV для подальшого їх використання у створенні моделей дерев рішень. Після валідації та фільтрації даних з двохсот опитувань залишилося сто сімдесят три.

3.3 Створення моделей дерев рішень

Для побудови класифікаційного і регресійного дерев рішень було вирішено використати мову програмування Python з використанням бібліотек для машинного

навчання. Спочатку було вичитано данні, на основі яких і будуть будуватися дерева. Приклад коду для вичитування даних в пам’ять наведено на рисунку 3.8.

```
# вичитуємо від валідовані дані
input_data = pd.read_csv("./validated_data.csv")
```

Рисунок 3.8 – Вичитування даних для побудови дерев

Для вичитування даних використовувалася бібліотека Pandas, яка має великий функціонал у роботі з даними. В результаті, вхідний набір даних було збережено у змінну input_data, де вони зберігалися у форматі DataFrame.

Наступним кроком було розбиття вхідних даних на атрибути та дані, які потрібно передбачити. Оскільки дерева рішень повинні передбачати схильність студента до ризику, потрібно згенерувати масив атрибутів без змінної передбачення. Для цього скористуємося функціоналом бібліотеки Pandas та видалимо колонку «Результати тесту» (див. рис. 3.9).

```
# створюємо дані для навчання
X_data = input_data.drop(['Результат тесту'], axis=1)
```

Рисунок 3.9 – Створення масиву атрибутів

Для масиву атрибутів скористалися методом drop, якому першим параметром передали масив з назвою колонки, яку потрібно видалити, і другим – координату осі. Координата осі вказує, що потрібно видалити. Допускаються два можливих значення – нуль і одиниця. Нуль відповідає за рядочок, а одиниця за колонку. Оскільки потрібно видалити колонку, а не рядочок, то у значення передали одиницю. Загальний результат зберегли до змінної X_data для подальшого використання.

Наступним кроком була генерація набору даних, які будуть передбачатися. Для цього потрібно отримати масив значень колонки «Результат тесту». Приклад коду наведено на рисунку 3.10.

```
# створюємо дані які будемо передбачати
Y_data = input_data['Результат тесту']
```

Рисунок 3.10 – Створення масиву даних для передбачення

Результат у вигляді структури даних Series було записано до змінної Y_data для подальшого використання.

Наступний кроком було виконано розбиття даних на навчальні та тестові. Навчальні дані – це дані, за якими будуть навчатися дерева рішень. Навчання або тренування – це процес, під час якого знаходяться логічні закономірності та створюється алгоритмічна модель, яка, основуючись на попередні дані, може передбачати або зіставляти вхідні дані.

Для спрощення логіки розбиття було використано об’єкт бібліотеки sklearn, який називається train_test_split. Цей об’єкт на вхід приймає набір даних атрибутів, набір цільових даних, розмір тестових даних (test_size) та випадковий стан (random_state). Розмір тестових даних відповідає за те, на яке співвідношення буде поділено вибірку. Зазвичай, хорошою практикою є співвідношення одного до трьох, де один це кількість даних для тестування, а три, відповідно, дані для навчання моделей. Параметр «випадковий стан» відповідає за керування перемішуванням, застосованим до даних, перед самим розділення. Цей параметр є важливим, адже дані потрібно розподілити хаотичним чином, тим самим гарантуючи кращий набір даних для навчання. На виході створюються чотири змінних, які, відповідно, містять набір атрибутів моделей для навчання, набір атрибутів моделей для тестування, набір цільових змінних для навчання та набір цільових змінних для тестування. Приклад коду наведено на рисунку 3.11.

Для розміру тестової вибірки було використано співвідношення один до трьох, а для випадкового стану вибрано число вісімдесят два, яке було отримано випадковим чином.

```
# розбиваємо на тестову і перевірочну вибірки
X_train, X_test, Y_train, Y_test = \
    train_test_split(X_data, Y_data, test_size=1/3, random_state=82)
```

Рисунок 3.11 – Розбиття даних для навчання та тестування

У змінній Y_train та Y_test зберігаються значення від нуля до сорока відповідно схильності до ризику студента. Така форма передбачуваних значень є хорошою для регресійної моделі, де передбачаються числові коефіцієнти, але для алгоритму класифікації ці значення є не коректними, адже алгоритм класифікації робить те, що розподіляє моделі за відповідними категоріями за атрибутами цих моделей. У випадку з результатами тестування є майже сорок класів для сто сімдесяти трьох моделей. В такому випадку, класифікаційне дерево рішень не зможе виконати свою задачу вірно. З точки зору логіки, дані для передбачення не є класами, адже числові коефіцієнти не класифікують студента за рівнем схильності до ризику.

У якості рішення цієї проблеми було вирішено конвертувати числові коефіцієнти в конкретні класи або групи. Згідно з методикою Шмельова, за числовими коефіцієнтами можна розподілити студентів на три групи. До першої групи відносяться студенти, у яких коефіцієнт схильності до ризику знаходиться від нуля до десяти. До другої групи відносяться студенти, у яких коефіцієнт схильності до ризику знаходиться від одинадцяти до тридцяти. І третя група студентів – від тридцяти одного до сорока.

Згідно з аналізом даних, було вирішено створити окремі змінні для зберігання цільових значень як для класифікаційного, так і для регресійного алгоритму. Для цього змінні, які містять цільові набори даних для тестування та передбачення, було скопійовано у відповідні до алгоритму змінні. Так для

класифікаційного алгоритму було створено змінні Y_train_class та Y_test_class, а для регресійного – Y_train_reg та Y_test_reg. Реалізацію цієї логіки наведено на рисунку 3.12.

```
# створення змінних передбачування
# для класифікаційної моделі
Y_train_class = Y_train.copy()
Y_test_class = Y_test.copy()

# створення змінних передбачування
# для класифікаційної моделі
Y_train_reg = Y_train.copy()
Y_test_reg = Y_test.copy()
```

Рисунок 3.12 – Копіювання даних передбачування

Змінні, які було отримано, знаходяться у формату структури даних Series, що допомагає обробляти їх за допомогою бібліотеки Pandas.

Далі дані для передбачування, які знаходяться у змінній Y_train_class та Y_test_class, перетворено на відповідні до психологічного тесту групи, де результат тестування між нулем та десяти відповідає низькому рівню схильності, між одинадцяти та тридцяти – середньому, а від тридцяти одного до сорока відповідає високому. Для спрощення було замінено низький, середній та високий на нуль, один та два відповідно.

Для програмної реалізації цієї логіки було використано метод loc, який, за допомогою логічного виразу, присвоює змінній той чи інший результат. Приклад логіки перетворення коефіцієнтів схильності до ризику у класи наведено на рисунку 3.13.

На даному етапі було отримано окремий масив з атрибутами моделей для тестування на навчання, за якими дерева рішень будуть передбачати схильність до ризику та масиви цільових даних, які розбитті для навчання та тестування. Також масиви цільових даних розбиті відповідно до алгоритму дерев рішень. Було

створено окрему групу цільових змінних для регресійного дерева рішень та окрему групу для класифікаційних дерев.

```
# перетворення цифрових коефіцієнтів у класи
Y_train_class.loc[np.logical_and(0 < Y_train_class, Y_train_class <= 10)] = 0
Y_train_class.loc[np.logical_and(10 < Y_train_class, Y_train_class <= 30)] = 1
Y_train_class.loc[np.logical_and(30 < Y_train_class, Y_train_class <= 40)] = 2

Y_test_class.loc[np.logical_and(0 < Y_test_class, Y_test_class <= 10)] = 0
Y_test_class.loc[np.logical_and(10 < Y_test_class, Y_test_class <= 30)] = 1
Y_test_class.loc[np.logical_and(30 < Y_test_class, Y_test_class <= 40)] = 2
```

Рисунок 3.13 – Перетворення цифрових коефіцієнтів у класи

Оскільки на цьому етапі уже створено набори даних для навчання та тестування, було вирішено переходити до побудови дерев рішень. Але, перш за все, потрібно було зрозуміти, які параметри найкраще підходять для класифікаційного та регресійного дерев. Адже необхідно зрозуміти, при якому алгоритмі розбиття (наприклад, для класифікаційного дерева рішень – це ентропія або коефіцієнт джині, а для регресійного дерева рішень – це середньоквадратична помилка) середня абсолютна похибка та розподіл Пуассона матимуть найкращий результат передбачення.

Також великого значення мають такі параметри дерев рішень, як глибина дерева. Глибина – це параметер, який показує, наскільки глибоко дерево рішень було побудоване. Цей параметер є важливим, так як він відповідає за перенавчання та неповне навчання дерев. Напркилад, якщо дерево перенавчилося, тобто створило окремий шлях для кожного цільового атрибута, то в такому випадку дерево не зможе точно передбачати інші дані, крім тих, на яких воно навчалося, адже воно не знайшло загальних закономірностей, за якими потрібно робити передбачення або розподілення, а просто побудувало закономірності для кожної цільової змінної. На противагу перенавчанню може статися недонавчання. Це може трапится, якщо ми скажемо алгоритму, наприклад, побудувати дерево рішень не глибше одного, а оптимальна модель, яка б могла знайти закономірності, буде десять. У такому

випадку, дерево не встигло знайти закономірності і буде робити передбачення неправильно.

Для того, щоб знайти дерево рішень, яке краще всього може бути побудованим на конкретному наборі даних, було використано клас GridSearchCV з бібліотеки sklearn. Але GridSearchCV – це не просто клас з бібліотеки. Це поняття в машинному навчання, а точніше – це процес виконання налаштування гіперпараметрів з метою визначення оптимальних значень для даної моделі. Як зазначалося вище, продуктивність моделі суттєво залежить від значення гіперпараметрів. Зауважте, що немає можливості заздалегідь дізнатися найкращі значення для гіперпараметрів, тому в ідеалі нам потрібно спробувати всі можливі значення, щоб знайти оптимальні значення. Виконання цього вручну може зайняти значну кількість часу та ресурсів, тому ми використовуємо GridSearchCV для автоматизації налаштування гіперпараметрів.

Для того, щоб користуватися GridSearchCV, потрібно передати в нього параметри дерев рішень, за якими ми хочемо побудувати оптимальні моделі. Приклад параметрів для класифікаційного дерева рішень наведено на рисунку 3.14.

```
# масив даних для пошуку оптимального
# класифікаційного дерева рішень
class_parameters = {
                    'criterion': ['entropy', 'gini'],
                    'max_depth': range(1, 50),
                    'min_samples_split': range(2, 10, 2),
                    'min_samples_leaf': range(1, 10, 2)
                    }
```

Рисунок 3.14 – Параметри для побудови оптимального класифікаційного дерева рішень

З наведеного прикладу можна побачити, що в якості критерію розбиття буде обиратися ентропія та коефіцієнт Джині. Для глибини дерева обрано проміжок від

одного до п’ятдесяти, так як вхідна кількість атрибутів є сорок чотири. В якості мінімальної кількості зразків, необхідних для розділення внутрішнього вузла, обрано проміжок від двох до десяти з кроком два. Для мінімальної кількість зразків, яка повинна знаходитися на листовому вузлі, обрано проміжок від одного до десяти з кроком два.

Також потрібно створити аналогічні параметри для регресійного дерева рішень. На рисунку 3.15 наведено атрибути для знаходження регресійного дерева рішень.

```
# масив даних для пошуку оптимального
# регресійного дерева рішень
reg_parameters = {
                    'criterion': ['mse', 'mae', 'poisson'],
                    'max_depth': range(1, 50),
                    'min_samples_split': range(2, 10, 2),
                    'min_samples_leaf': range(1, 10, 2)
                }
```

Рисунок 3.15 – Параметри для побудови оптимального регресійного дерева рішень

З наведеного прикладу можна побачити, що в якості критерію розбиття буде обиратися середньоквадратична помилка, середня абсолютна похибка та розподіл Пуассона. Для глибини дерева обрано проміжок від одного до п’ятдесяти, так як вхідна кількість атрибутів є сорок чотири. В якості мінімальної кількості зразків, необхідних для розділення внутрішнього вузла, обрано проміжок від двох до десяти з кроком два. Для мінімальної кількості зразків, яка повинна знаходитися на листовому вузлі, обрано проміжок від одного до десяти з кроком два.

Під час наступного кроку було знайдено оптимальний набір параметрів для дерев рішень для класифікаційного та регресійного алгоритму. Аналіз проводився

за допомогою об’єкту GridSearchCV. Для цього, перш за все, було створено об’єкти відповідних моделей, які містить бібліотека sklearn (див. рис. 3.16).

```
# створення об'єкту класифікаційного дерева рішень
class_tree = tree.DecisionTreeClassifier()

# створення об'єкту регресійного дерева рішень
reg_tree = tree.DecisionTreeRegressor()
```

Рисунок 3.16 – Створення об’єктів класифікаційного та регресійного дерев рішень

Наступний крок – це створення GridSearchCV для конкретних моделей дерев рішень із відповідними їм параметрами. GridSearchCV на вхід першим параметром приймає дерево рішень відповідного алгоритму, другим параметром приймає параметри, за якими буде проходити побудова та перевірка дерева рішень, а третім параметром задається, на скільки наборів даних буде розбита вибірка для навчання для проходження кросвалідації. Кросвалідація, яку іноді називають перехресною перевіркою, – це техніка валідації моделі для перевірки того, наскільки успішно застосовується в моделі статистичний аналіз здатний працювати на незалежному наборі даних. Зазвичай кросвалідація використовується в ситуаціях, де метою є передбачення, і необхідно оцінити, наскільки модель передбачення здатна працювати на реальних даних. Один цикл кросвалідації включає розбиття набору даних на частини, потім побудова моделі на одній частині (званої тренувальним набором), і валідація моделі на іншій частині (званої тестовим набором). Щоб зменшити розкид результатів, різні цикли кросвалідації проводяться на різних розбиттях, а результати валідації усереднюються по всіх циклах. В середньому для кросвалідації дані розбиваються на п’ять частин, але оскільки у якості набору даних для тренування є масив невеликого розміру, то для кращого тестування у якості коуфіцієнта кросвалідації було використано цифру чотрити. На рисунку 3.17

показано створення GridSearchCV-об’єктів для класифікаційного та регресійного дерев рішень.

```
# створення об'єкту GridSearchCV для класифікаційного дерева рішень
class_grid_search = GridSearchCV(class_tree, class_parameters, cv=4)

# створення об'єкту GridSearchCV для класифікаційного дерева рішень
reg_grid_search = GridSearchCV(reg_tree, reg_parameters, cv=4)
```

Рисунок 3.17 – Створення об’єктів GridSearchCV для класифікаційного та регресійного дерев рішень

Наступним кроком був запуск алгоритму для пошуку оптимальних параметрів для кожного з дерев на даних для навчання. Для цього було використано метод fit об’єкту GridSearchCV і виконано його для класифікаційного дерева рішень з використанням змінної, яка містить цільові значення для тренування класифікаційного дерева рішень. Виклик функції fit відображено на рисунку 3.18.

```
# знаходження найркащого набору параметрів
# для класифікаційного дерева
class_grid_search.fit(X_train, Y_train_class)
```

Рисунок 3.18 – Знаходження найкращого набору параметрів для класифікаційного дерева

У результаті було отримано найкращу комбінацію параметрів для класифікаційного дерева та навчено дерево рішень із використанням цих параметрів. На рисунку 3.20 відображено найкращу комбінацію параметрів для класифікаційного дерева рішень.

Як видно з прикладу, оптимальним дерево рішень є побудованим із використанням критерією розбиття ентропією, максимальною глибиною чотири,

мінімальною кількістю зразків для розподілення та на листу чотири та один відповідно.

```
{'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 4}
```

Рисунок 3.20 – Найкраща комбінація параметрів для класифікаційного дерева рішень

За допомогою бібліотеки Graphviz було візуалізовано класифікаційне дерево рішень, яке зображено на рисунку 3.21.

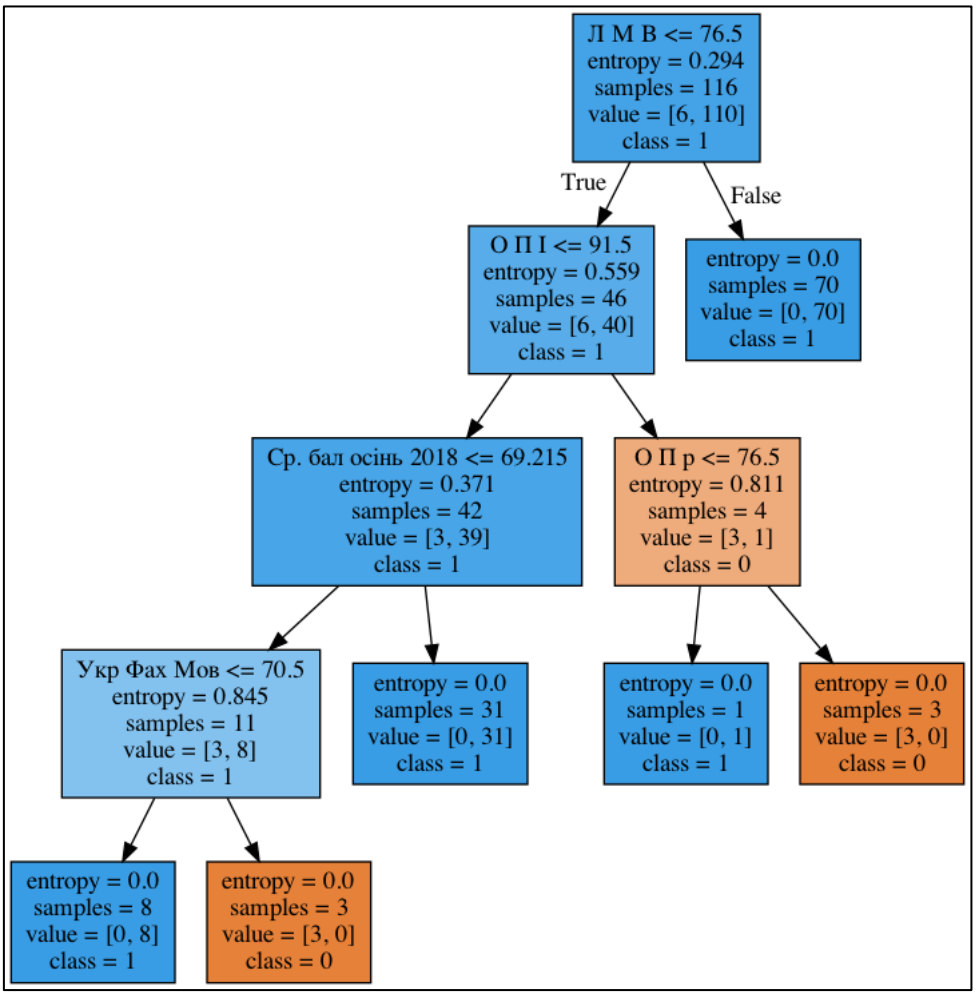

Рисунок 3.21 – Класифікаційне дерево рішень

Наступним кроком було проведено пошук оптимального регресійного дерева рішень. Для цього на об’єкті reg_grid_search викликано метод fit з використанням

змінної, яка містить цільові значення для тренування регресійного дерева рішень як показано на рисунку 3.22.

```
# знаходження найкращого набору параметрів
# для регресійного дерева
reg_grid_search.fit(X_train, Y_train_reg)
```

Рисунок 3.22 – Знаходження найкращого набору параметрів для класифікаційного дерева

На рисунку 3.23 відображена найкраща комбінація параметрів для регресійного дерева рішень.

```
{'criterion': 'mae', 'max_depth': 1, 'min_samples_leaf': 1, 'min_samples_split': 6}
```

Рисунок 3.23 – Найкраща комбінація параметрів для регресійного дерева рішень

За допомогою бібліотеки Graphviz було візуалізовано регресійне дерево рішень, яке зображено на рисунку 3.24.

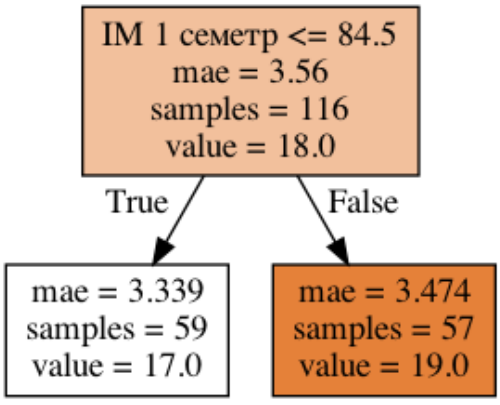

Рисунок 3.24 – Регресійне дерево рішень

У підсумку, було побудовано два оптимальних дерев рішень для класифікаційного та регресійного алгритмів.

4 ПОРІВНЯННЯ ДЕРЕВ

3.4 Час навчання

Користуючись бібліотекою time, було заміряно час навчання для класифікаційного та регресійного дерев рішень. Результати замірів наведено в таблиці 4.1.

Таблиця 4.1 – Результати тестування на час навчання

№ тесту	Класифікаційне дерево рішень (в мілісекундах)	Регресійне дерево рішень (в мілісекундах)
1	2.45	5.45
2	2.43	5.43
3	2.17	5.04
4	2.42	5.53
5	2.37	5.36
6	2.21	4.99
7	2.52	5.51
8	2.57	5.61
9	2.36	5.41
10	2.79	6.00
Середнє	2.43	5.43

Навчання дерев проводилося на класифікаційному дереві рішень з критерієм розбиття ентропією, максимальною глибиною чотири, мінімальною кількістю зразків для розподілення та на листу чотири та один відповідно. Для регресійного дерева рішень використовувалися критерій розбиття середня абсолютна похибка, максимальна глибина один, мінімальна кількість зразків для розподілення та на листу шість та один відповідно. Усі тести проводилися використовуючи мову програмування Python 3.8. В якості операційної системи використовувалася macOS

Catalina версії 10.15.7. Тестування проводилося на машині з процесором 2 GHz Quad-Core Intel Core i5 та оперативною пам’яттю 16 гігабайт.

Опираючись на результати тестування можна сказати, що класифікаційний алгоритм навчається майже в два з половиною рази швидше, ніж регресійний, незважаючи на те, що регресійний алгоритм має глибину один.

3.5 Метрики якості

Побудовані класифікаційні та регресійні дерева рішень були проаналізовані на точність передбачення. Для цього були отримані такі метрики для класифікаційного дерева рішень, як точність, повнота та акуратність. Для регресійного дерева рішень отримувалася метрика відсоткова середня абсолютна похибка. Для отримання цих метрик програмним шляхом було використано модуль metrics в бібліотеці sklearn.

Точність (precision англ.) – це частка правильно передбачених значень серед правильно передбачених та захоплених значень, які були передбачиними як правильні, але насправді такими не являються. Точність – це метрика, яка показує стійкість дерева рішень до помилок першого роду. Точність вираховується за формулою (4.1).

$$Precision = \frac{TP}{TP + FP}, \tag{4.1}$$

де TP – це кількість значень, які передбачені як правильні і які насправді являються правильними, FP – це передбачення значень, які передбачені як правильні і які насправді є неправильними. Значення точності можуть знаходитися в диапазоні від нуля до одиниці, де нуль – це алгоритм, не стійкий до помилок першого роду, а один, відповідно, стійкий.

Повнота (recall англ.) – це частка правильно передбачених серед правильно передбачених та захоплених значень, які були передбачиними як неправильні, але насправді є правильними. Точність – це метрика, яка показує стійкість дерева рішень до помилок другого роду. Повнота вираховується за формулою (4.2).

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

де FN – це передбачення значень, які передбачені як неправильні і які насправді є правильними. Значення повноти можуть знаходитися в диапазоні від нуля до одиниці, де нуль – це алгоритм, не стійкий до помилок другого роду, а один, відповідно, стійкий.

Точність та повнота характеризують, наскільки стійким є аглоритм класифікаційного дерева до помилок першого та другого роду. Ці метрики не можуть йти як незалежні показники, адже вони не показують, наскільки дерево точне у своїх передбаченнях.

Акуратність (accuracy англ.) – це метрика, яка показує, наскільки точно класифікаційне дерево рішень може передбачати результат.

Виходячи з акуратності, можна вирахувати, наскільки класифікаційне дерево рішень є неточним у своїх передбачень. Це можна вирахувати за формулою (4.3).

$$Error = (1 - Accuracy) * 100\% \tag{4.3}$$

Для вирахування точності та повноти було використано об’єкти precision_score та recall_score модуля metrics. В результаті, було отримано, що точність класифікаційного дерева є 0.96 та повнота – 0.96. Відповідно, було зроблено висновок, що дерево рішень є стійким до помилок першого та другого роду.

Щоб вирахувати акуратність, було використано об’єкт accuracy_score модуля metrics. В результаті було отримано значення 0.93. Відповідно до формули (4.3)

було вирахувано відсоток помилок для класифікаційного дерева рішень, що дорівнює семи відсоткам.

Для регресійного дерева була обрано відсоткова середня абсолютна похибка. Ця метрика показує, на скільки відсотків регресійне дерево рішень робить передбачення з помилками. Відсоткова середня абсолютна похибка вираховується за формулою (4.4).

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\frac{|y_t - \widehat{y_t}|}{y_t}, \tag{4.4}$$

де y_t – цільове значення, $\widehat{y_t}$ – передбачене значення. Для вирахування обраної відсоткової середньої абсолютної похибки було використано об’єкт mean_absolute_percentage_error модуля metrics. Значення цієї похибки для регресійного дерева рішень дорівнює двадцять п’ять відсотків.

Посилаючись на отримані метрики, було зроблено висновок, що класифікаційне дерево рішень є стійкішим та робить передбачення з більшою точністю, ніж регресійне.

ВИСНОВОК

У ході даної роботи було проаналізовано такі алгоритми машинного навчання, як дерева рішень, лінійна регресія та нейронні мережі, та описано, які переваги та недоліки має кожен з них і в якій області їх краще використовувати.

Було проаналізовано проблему у використанні алгоритмів машинного навчання у програмних додатках та описано, з якими викликами зіштовхуються розробники цих програмних додатків. На основі цього аналізу було поставлено задачу дослідити ефективність застосування дерев рішень для обробки результатів психологічного тестування.

За основу психологічного тестування було взято методику «Дослідження схильності до ризику», автором якої є О. Г. Шмельов.

Також в процесі даної роботи було проведено дослідження побудови дерев рішень та описано головні підходи та труднощі, з якими зіштовхуються при побудові дерев.

В результаті дослідження було згенеровано набір даних із психологічного тесту студентів та їх оцінок. Ці дані були відвалідовані та перетворені для того, щоб побудувати дерева рішень.

Було знайдено оптимальний набір параметрів для побулов дерев рішень класифікаційного та регресійного алгоритмів на основі згенерованого набору даних. На основі цих параметрів були побудовані класифікаційне та регресійне дерева рішень.

Дерева рішень були проаналізовані та порівняні, використовуючи метрики точності, повноти, акуратності та відсоткової середньої абсолютної похибки.

У результаті було зроблено висновок, що для тренування регресійного дерева рішень потрібно майже в два з половиною рази більше часу. За результатами порівняння дерев було визначено, що класифікаційне дерево рішень дає більш точний прогноз, ніж регресійне дерево рішень. Класифікаційне дерево рішень помиляється у семи відотках випадків, коли регресійне – у двадцяти п’яти.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Шмелев А. Г. Психодиагностика личностных черт, 2002. – 472 с.
2. Xin Y. Linear Regression Analysis: Theory and Computing, 2009. – 327 с.
3. Schmidhuber J. Deep learning in neural networks: An overview. URL: https://doi.org/10.1016%2Fj.neunet.2014.09.003 (дата звернення 27.03.2021).
4. Smelyakov K., Pribylnov D., Martovytskyi V., Chupryna A. Investigation of network infrastructure control parameters for effective intellectual analysis // IEEE 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering, 20-24 Feb. – 2018. – P. 983-986.
5. Smelyakov K., Yeremenko D., Sakhon A., Polezhai V., Chupryna A. Braille character recognition based on neural networks // IEEE Second International Conference on Data Stream Mining & Processing (DSMP), August 21-25. – 2018. – P. 509-513.
6. Kamiński, B. A framework for sensitivity analysis of decision trees – Hamburg: Central European Journal of Operations Research, 2018.
7. Shital K. Logistic Regression vs. Decision Tree. URL: https://dzone.com/articles/logistic-regression-vs-decision-tree (дата звернення 29.03.2021).
8. Cordoni F. A comparison of modern deep neural network architectures for energy spot price forecasting. URL: https://link.springer.com/article/10.1007/s42521-020-00022-2 (дата звернення 30.03.2021).
9. Classification and regression tree analysis in public health: Methodological review and comparison with logistic regression. URL: https://link.springer.com/article/10.1207/S15324796ABM2603_02 (дата звернення 02.04.2021).
10. Hovland C. Computer simulation of thinking, 1960. – 693 с.
11. Hunt. Experiments in Induction – New York: Academi, 1966.
12. Quinlan J. Induction of decision trees. Machine Learning, 1986. – 106 с.
13. Quinlan, J. Ross. Programs for Machine learning – Burlington: Morgan Kaufmann Publishers, 1993.