ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ ТА МАШИННЕ НАВЧАННЯ
ЧАСТИНА 1
БАЗОВІ МЕТОДИ ТА ЗАСОБИ АНАЛІЗУ ДАНИХ

Міністерство освіти і науки України
Вінницький національний технічний університет

ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ ТА МАШИННЕ НАВЧАННЯ
ЧАСТИНА 1
БАЗОВІ МЕТОДИ ТА ЗАСОБИ АНАЛІЗУ ДАНИХ

Навчальний посібник

Вінниця
ВНТУ
2021

УДК 004.652
І86

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 15 від 31.05.2021 р.)

Автори:
Я. В. Іванчук, В. І. Месюра, А. А. Яровий, О. Д. Манжілевський

Рецензенти:
Р. Н. Квєтний, доктор технічних наук, професор
Т. Б. Мартинюк, доктор технічних наук, професор
І. Г. Цмоць, доктор технічних наук, професор

І86 **Інтелектуальний** аналіз даних та машинне навчання. Частина 1. Базові методи та засоби аналізу даних / Я. В. Іванчук, В. І. Месюра, А. А. Яровий, О. Д. Манжілевський – Вінниця : ВНТУ, 2021. – 69 с.
ISBN 978-966-641-874-9

Посібник містить теоретичний матеріал і приклади розв'язування практичних задач з інтелектуального аналізу даних; цілі, практичні завдання, переліки контрольних питань та вимоги до знань студентів, потрібні для виконання лабораторних робіт, що стосуються першого модуля навчальної дисципліни «Інтелектуальний аналіз даних та машинне навчання2.

Розрахований на студентів комп'ютерних спеціальностей всіх форм навчання.

УДК 004.652

ISBN 978-966-641-874-9

ЗМІСТ

ВСТУП

Навчальний посібник призначений для студентів спеціальності 122 – «Комп'ютерні науки». Зміст навчального посібника відповідає плану і програмі дисципліни «Інтелектуальний аналіз даних та машинне навчання».

Аналіз даних – сфера математики та інформаційних технологій, яка займається побудовою і дослідженням математичних методів і обчислювальних алгоритмів отримання бази знань із експериментальних даних [1–3]. Дана сфера охоплює процеси збирання, структурування і моделювання даних, їх дослідження та фільтрації з метою отримання корисної інформації та прийняття рішень. Математичні методи побудови моделей на основі даних об'єднуються в науку, яка називається машинним навчанням [4]. Сукупність наук, спрямованих на методи і технології роботи з даними, називають Data Science [1–7].

У зв'язку зі стрімким розвитком інформаційних технологій за останні десятиліття дані природним чином стали накопичуватися в цифровому вигляді, і тому наукова галузь інтелектуального аналізу даних стала активно розвиватися. До теперішнього час накопичені дані досягли значних обсягів, за допомогою яких з'явилася можливість отримувати з них користь: знаходити в даних приховані залежності, а з їх допомогою прогнозувати нові результати і робити рекомендації, що дозволяють оптимізувати різні процеси і витрати ресурсів [8]. Методи аналізу даних і машинного навчання активно розвиваються і знаходять своє застосування не тільки в економіці, фізиці, але й у соціальних науках, журналістиці, лінгвістиці, юриспруденції, політології та гуманітарних науках тощо [9].

Тому актуальним є поява навчального посібника з дисципліни «Інтелектуальний аналіз даних», який дозволить формувати у студентів компетенції, пов'язані з напрямом науки Data Science. Отримані навички дозволять у перспективі швидко та ефективно інтегруватися в розв'язання професійних задач на стику предметних галузей та комп'ютерних технологій, які сьогодні є передовими, але вже в найближчій перспективі стануть звичною практикою.

Перша частина навчального посібника містить потрібний теоретичний матеріал з базових методів і засобів аналізу даних в програмі PYTHON. Також даний навчальний посібник містить велику кількість прикладів розв'язування практичних задач інтелектуального аналізу даних за допомогою вбудованих бібліотек програми PYTHON; цілі, практичні завдання, переліки контрольних питань та вимоги до знань студентів, потрібні для виконання лабораторних робіт, що віднесені, згідно з навчальною програмою, до першого модуля дисципліни «Інтелектуальний аналіз даних та машинне навчання».

1 МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ТА АНАЛІЗ ДАНИХ ПРОГРАМНИМИ ЗАСОБАМИ PYTHON

1.1 Засоби роботи з багатовимірними масивами даних бібліотеки NumPy

Дана бібліотека призначена для різного роду математичних обчислень і роботи з багатовимірними масивами [1]. Вона досить потужна, має великий математичний потенціал, гнучкий і зрозумілий інтерфейс. Основна структура NumPy – це певний багатовимірний масив, який створюється на основі отриманих даних. Для створення деякого масиву потрібно імпортувати дану бібліотеку, після чого викликати функцію array, яка приймає на вхід дані у вигляді деякої послідовності

import numpy as np

x = np.array([1,2,3,4]).

За допомогою команди *x.dtype* можна визначити, які дані зберігаються в масиві. Тип даних може визначатися автоматично на основі того, які є вхідні дані, а також можна його вказувати при створенні масиву як додатковий аргумент

x = np.array([1,2,3,4], *dtype=np.int*64).

У даному випадку було створено певний одновимірний масив і для визначення розмірності потрібно використати команду *x.shape*. Атрибут *shape* повертає набір розмірності по всіх осях, які у нас є в масиві. Тобто, при двовимірному масиві було б отримано пару чисел, а саме: кількість рядків і число стовпців, але в даному випадку для одновимірного масиву отримують тільки одне значення.

Для створення двовимірного масиву потрібно ввести

m = np.array([[2,3,4], [5,6,7]]).

За допомогою NumPy масиви можна створювати декількома способами. Ми розглянули метод створення масиву на основі *list*, також можна спробувати створити масиви на основі вбудованої функції. Припустимо, якщо потрібно отримати масив, заповнений тільки одиницями, то можна скористатися функцією *ones*, яка як аргумент приймає розмір нашого масиву, в результаті ми отримуємо вже масив, заповнений тільки одиницями,

m = np.ones(5).

Таким же способом можна створити масив, заповнений тільки нулями,

m = np.zeros(2).

Також можна створити деяку одиничну матрицю, де по діагоналі будуть розміщені одиниці, а інші значення будуть нульовими

m = np.eye(6).

Для створення масиву, заповненого випадковими значеннями, можна скористатися методом *random*

*m = np.random.randint(*10, *size* = (2, 3)).

Після того, як було створено масив, потрібно з ним взаємодіяти. Щоб визначити якісь певні значення елементу масиву, потрібно, вибравши потрібний нам індекс, звернутися за ним і отримати значення (індексація починається з нуля)

x[0].

У свою чергу, для двовимірної матриці потрібно задати індекси відразу по двох осях, тобто спочатку першим аргументом в квадратних дужках вказується індекс по осі *X*, а як другий аргумент вказуємо індекс по осі *Y*

m[0,3].

Можна вказувати також деякий проміжок значення індексу: обираються усі значення по рядках і обираються тільки значення тільки перших трьох стовпців

m[:,:3].

Також можна отримувати значення, які задовольняють певну умову. Наприклад, для отримання усіх значень, що перевищують будь-яке задане число, для початку потрібно задати наші умови, в результаті чого ми отримуємо деяку маску. Маска – це масив того ж розміру, що й наш вихідний, але як значення вказується *true* або *false*, що означає, що дане число в масиві перевищує, або не перевищує задану межу. Після чого, щоб отримати потрібне значення, дана маска передається як індекс в наш масив, в результаті чого повертається вже масив значень, що відповідає умові

x[*x* > 2].

При роботі з багатовимірними масивами виникає ситуація, що потрібно змінити форму, або розмірність даних. Наприклад, для багатовимірної структури виконати розгортання в один масив (одновимірний), і прогляну-ти деякі значення. У NumPy це можна здійснити, використовуючи методи *flatten* і *reshape*. Для розгортання усіх значень, що зберігаються в нашій матриці, в одновимірний масив виконуємо

x = *np.array*([[1,2,3],[6,5,4]])

x.flatten().

Використовуючи метод *reshape* можна отримати деякий масив з цих елементів інших розмірів, наприклад по двох осях *X* і *Y*, які ми хочемо побачити,

x.reshape((6, 1)).

Також, використовуючи метод *resize*, можна змінити форму нашого вихідного масиву. Відмінність *resize* від *reshape* полягає в тому, що *resize* автоматично змінює вихідний масив, в той час як *reshape* просто змінює його форму

x.resize((6, 1)).

В NumPy над векторами можна виконувати такі математичні операції, як поелементне додавання, множення, віднімання:

v = *np.array*([9,10])

w = *np.array*([11,12])

res = *v* + *w*

res = np.add(v, w)
*res = v * w*
res = np.multiply(v, w)
res = v - w
res = np.subtract(v, w).

Для того, щоб отримати скалярний добуток векторів, або вектора та матриці, потрібно скористатися методом *dot*

res = np.dot(v, w).

1.2 Засоби реалізації математичних операцій в бібліотеці SciPy

Дана бібліотека призначена для виконання широкого спектра математичних операцій та охоплює: методи оптимізації, методи лінійної алгебри, обробки сигналів і зображень [1, 2].

Визначимо детермінант заданої матриці, але перед цим виконаємо імпорт відповідних модулів із ScyPi:

from scipy import linalg
from scipy import optimize
import numpy as np
import matplotlib.pyplot as plt
A = np.array([[1,3,5],[2,5,1],[2,3,8]])
linalg.det(*A*),

але потрібно зауважити, що детермінант визначається тільки для квадратних матриць.

Визначення оберненої матриці:

linalg.inv(*A*).

Для визначення власних чисел і векторів в бібліотеці SciPy реалізований метод, який визначає пару чисел, де першим значенням будуть власні числа, а другим – власні вектори:

e*igenvalues*, *eigenvectors = linalg.eig*(*A*).

У підсумку повертається масив, в якому знаходяться власні числа, а також повертається матриця, в якій знаходяться власні вектори.

Задамо нижченаведену функцію:

def f(*x*):
*return x***2 + 6**np.sin*(*x*).

Визначимо похідну даної функції в точці $x = 1$, для цього імпортуємо модуль *derivative*

from scipy.misc import derivative
derivative(*f*, 1.0, *dx*=1e-6),

де одним із аргументів методу *derivative* є параметр точності *dx*, який відповідає за кількість знаків після коми.

Визначимо первісну даної функції в точці $x = 1$, для цього імпортуємо модуль *quad*

from scipy.integrate import quad

quad(*f*, 0, 1),

де пара чисел 0 і 1, що вказана в дужках, є межами інтегрування.

Побудова графіка даної функції:

x = np.arange(-10, 10, 0.1)

f2=np.vectorize(*f*)

plt.plot(*x, f2*(*x*))

plt.show().

Аналізуючи даний графік визначаємо, що глобальний мінімум знаходиться в околі -1. Для точного обчислення мінімуму функції в SciPy є відповідний метод *minimize*. В опції даного методу як аргумент передається наша функція, а також початкове наближення, тобто область, з якої починається пошук мінімуму (для прикладу приймаємо 0).

optimize.minimize(*f*, 0).

Результатом роботи функції буде деякий об'єкт, в якому присутні безліч параметрів. Багато з цих параметрів належать до обраного методу оптимізації, а останній буде відповідати мінімуму функції. Потрібно відмітити, що метод повернув за мінімум значення -1, що відповідає глобальному мінімуму.

Важливо відзначити, що метод *minimize* має більше число аргументів, ніж було вказано. Можна також вказувати метод оптимізації [2] залежно від того, яка у нас функція, а також точність наближення, з якої ми хочемо знайти мінімум.

Метод *minimize* також може помилятися. Зокрема, якщо у нас є локальний мінімум, тоді випадково замість глобального мінімуму метод може повернути локальний мінімум, і цього легко добитися, змінивши лише один параметр, наприклад, початкове наближення. Наприклад, задаючи значення початкового наближення рівне трьом, тоді результат, який відповідає мінімуму, в даному випадку наближається до значення 3, що схоже на один з локальних мінімумів функції.

1.3 Засоби роботи із подготовкою та аналізом даних у бібліотеці Pandas

Бібліотека Pandas є однією з ключових бібліотек [1, 3], яка дозволяє зробити повноцінний процес аналізу даних, а саме:

– завантаження даних. Бібліотека Pandas дозволяє завантажувати дані різних форматів. Бібліотека підтримує завантаження текстових файлів, бінарних, а також є можливість підключення до баз даних і роботи з ними напряму;

– подання даних. Pandas оперує двома основними структурами даних: Series і DataFrame. Series – індексований масив деякого типу: числовий, бінарний або категоріальний. DataFrame – двовимірна структура даних (сукупність Series) або – це таблиця;

– обробка даних. Після завантаження та формування Series або DataFrame виконання фільтрації, індексування, об'єднання різних

DataFrame один з одним, є можливість написання власних обчислювальних методів і застосовування їх на DataFrame;

– побудова графіків. За обраними даними з DataFrame можна будувати гістограми значень, можна подивитися як значення в даному стовпці змінюється в часі. Візуалізація можлива завдяки інтеграції Pandas з іншою бібліотекою – Matplotlib. Загалом Pandas інтегрується й з такими бібліотеками, як NumPy або SciPy.

1.3.1 Об'єкт Pandas.Series

Структура Series [4] є об'єктом, схожим на одновимірний масив, але його відмінною рисою є наявність міток, тобто індексів біля кожного елемента зі списку

import pandas as pd

s = pd.Series([1,2,3,4], ['*a*', '*b*', '*c*', '*d*']).

Для створення *Series* необов'язково вказувати індекси, Pandas автоматично проставить числовий індекс. Також *Series* можна створювати, передаючи не тільки лист з даними, а й словник, ключі якого стануть індексами в створеному *Series*:

d = {'*Moscow*': 1000, '*London*': 300, '*Barcelona*': *None*}

cities = pd.Series(*d*).

Отримати значення в *Series* можна за індексом

print (*cities*['*Moscow*']),

або за індексами

print cities['*Moscow*','*London*'];

або можна отримати значення за індексами

print(s['*a*' : '*c*']).

Для того, щоб із *Series* отримати деякі рядки, які задовольняють деякі умови, наприклад значення, що не перевищують певної межі, спочатку потрібно створити маску у формі умови, якій має відповідати *Series*, а потім передати її як індекс

print cities[*cities* < 1000].

Для зміни значення в *Series* достатньо вибрати значення в потрібному індексі й присвоїти йому нове

cities['*Moscow*'] = 100.

Для зміни значень в *Series*, що були відфільтрованими, наприклад, за заданою маскою, можна це зробити аналогічним способом

cities[*cities* < 1000] = 3.

Якщо в *Series* зберігаються числові значення, то можна простим способом змінити значення усього Series за допомогою арифметичних операцій

print(*cities*[*cities*>0]*3).

У наведеному *Series* присутні значення *NaN*, які відповідають тому, що за індексом «Барселона» у нас відсутні значення. Для відфільтрування типових рядків у Pandas реалізовано два методи *isnull* і *notnull*, які протилежні один одному,

print (*cities*[*cities.isnull*()]).

Метод *isnull* повернув нам тільки один рядок, але якщо ми спробуємо викликати метод *notnull*, тоді нам повернуться й усі інші рядки, в яких є значення

print (*cities*[*cities.notnull*()]).

1.3.2 Об'єкт Pandas. DataFrame

Для роботи з модулем DataFrame імпортуємо потрібні нам бібліотеки:

import pandas as pd

import numpy as np.

На прикладі датасету поїздок велобайком в Нью-Йорку

df = *pd.read_csv*('*citibike.csv*')

розглянемо, як влаштований сам *DataFrame*. Не потрібно роздруковувати весь *DataFrame*, коли ми хочемо дізнатися його структуру, які там містяться стовпці. Припустимо, що потрібно подивитися тільки на перші три рядки, тому в Pandas є метод *Head*, який за замовчуванням повертає перші п'ять рядків нашого DataFrame,

df.head(3).

Якщо потрібно подивитися не перші три рядки DataFrame, а останні, тоді можна скористатися методом *tail*, який, аналогічно, повертає останні п'ять рядків за замовчуванням,

df.tail(3).

Крім цього DataFrame має такі властивості, як визначення розміру (спочатку вказується число рядків, потім вказується число стовпців),

df.shape.

також є можливість отримання окремих назв стовпців

df.columns,

і які типи даних є в нашому DataFrame

df.dtypes.

Також є можливість виведення не всього DataFrame, а певних стовпців. Для цього потрібно вказати назви потрібних нам стовпців і передати їх, а тоді отримуємо можливість друкувати не весь DataFrame, а тільки обрані стовпці

df[['*start time*', '*start station name*']].*head*().

Для звернення до елементів DataFrame у Pandas реалізовані такі методи, як *loc* й *iloc*. *Iloc* дозволяє за індексом звертатися до рядків і стовпців. Зокрема, якщо потрібно отримати значення, що зберігаються в найостаннішому рядку, в *iloc* потрібно вказати індекс -1, і за замовчуванням повернеться останній рядок

df.iloc[-1].

Також за допомогою методу *iloc* можна подивитися, яке значення буде містити DataFrame на перетині стовпця і рядка,

df.iloc[-1, 4].

Метод *loc* відрізняється від *iloc* тим, що ми можемо вказувати значення зрізу за індексом,

df.loc[1,['*tripduration*']].

Якщо потрібно відфільтрувати за допомогою методу *iloc*, вказуючи межі, тоді у нас не вказуються крайні значення. Метод *loc* працює навпаки, він містить крайні значення. Для порівняння:

df.loc[0:6,0:4]

df.iloc[0:6, '*tripduration*':'*start duration time*'].

Найчастіше при роботі з DataFrame ми стикаємося з тим, що нам потрібно вибрати рядки, що задовольняють певні умови. Для цього потрібно створити маску, яка буде заданою умовою, й передати її як індекс нашому DataFrame. Також можна створювати декілька умов на фільтрацію

df[(*df*['*tripduration*'] < 1000) & (*df*['*usertype*'] = = '*Subscriber*')].

Окрім фільтрації в Pandas реалізовані деякі методи, які дозволяють подивитися статистику усього DataFrame. Наприклад, за допомогою методу *describe* можна подивитися середнє, максимальне і мінімальне значення одразу в усіх стовпцях

df.describe().

Оскільки наш DataFrame зберігає не тільки числові, але й категоріальні ознаки, то за допомогою методу *describe* за ними також можна визначити статистику. Для цього потрібно вказати, який тип даних ми хочемо отримати,

df.describe(*include*=[*np.object*]).

При роботі з категоріальним ознаками для отримання співвідношення деяких значень використовують метод *value_counts*

df['*usertype*'].*value_counts*(*normalize*=*True*).

Для отримання кількості унікальних значень в окремих *Series* використовують метод *unigue*

df['*gender*'].*unique*().

Також можна отримати кореляцію між усіма стовпцями датасету, якщо вони числові, використовуючи метод *corr*,

df.corr().

Крім того, буває потрібно зробити певний *sample* з вихідного DataFrame (DataFrame надто об'ємний і потрібно зберегти окрему частину). За допомогою методу *sample* виконується вибір певної частини з вихідного DataFrame

df.sample(*frac*=0.1).

Після зазначених перетворень можна зберегти DataFrame і використати метод *to_csv*. Для цього потрібно вказати шлях до файлу і вказати інші параметри

df.to_csv('*path_to_file.csv*').

1.4 Методи групування даних

З метою огляду методів групування та маніпуляцій даних в Pandas розглянемо деяку змінну [5], що може набувати два або кілька значень, і потрібно визначити всі рядки, в яких зустрічається тільки перше значення да-

ної змінної, потім – тільки друге і т. д. Це можна здійснити за допомогою методу *groupby*. Після того, як у нас виділилася певна група, з нею можна окремо працювати, аналізувати розподіл різних ознак у даній групі, а також можна порівнювати ці групи між собою [6]

```
import pandas as pd
import numpy as np

df = pd.read_csv('citibike.csv')
df.groupby(['usertype']).
```

Огляд груп, які було отримано:

```
df.groupby(['usertype']).groups.
```

У результаті буде отримано певний словник, де як ключ знаходяться значення зі згрупованою змінною. У даному випадку їх усього два, де за значення використовуються індекси рядків, в яких зустрічаються саме ці значення.

Окрім даної форми подання у вигляді індексів можна визначити, які саме значення і рядки у нас зустрічаються в групі. Для цього потрібно перевірити перші рядки з кожної групи, викликавши метод *first*,

```
df.groupby(['usertype']).first(),
```

при цьому отримуємо деякий DataFrame з усіма ознаками, які були спочатку.

На основі отриманої певної згрупованої структури можна порахувати деякий розподіл значень. Наприклад, потрібно визначити середню тривалість поїздок у кожній групі користувачів. Для цього здійснюється групування даних, вказується лист значень, потрібних для агрегування, а потім метод, за яким агрегуються дані, наприклад, середня тривалість поїздок,

```
df.groupby(['usertype'])[['tripduration']].mean(),
```

при цьому отримуємо деякий DataFrame, в якому як аргументи вказати не список значень, а усього лише одне значення (тривалість поїздки), тоді у нас вже повернеться об'єкт типу *Serious*.

Також Pandas дозволяє групувати дані не тільки за якоюсь однією ознакою, а й відразу за групою ознак. Для цього потрібно додати додатковий стовпець в метод *groupby*.

Після групування даних Pandas з'являється можливість обчислювати певне агреговане значення за ознаками для кожної групи. Але якщо потрібно визначити значення не тільки за однією ознакою, а й відразу за кількома ознаками, тоді потрібно використати метод *agg*

```
df.groupby(['usertype']).agg({'tripduration':sum, 'starttime':'first'}).
```

У результаті застосування даного методу повертається DataFrame, в якому індекс – це вихідна ознака групування й відповідні два стовпці з потрібними значеннями.

Більше того, якщо потрібно визначити зміну значень або взагалі будь-які інші метрики за якоюсь однією ознакою, то в даному словнику (для даної ознаки як значення) потрібно вказати список методів

df.groupby(['*usertype*']).*agg*({'*tripduration*':[*sum*, *min*], '*starttime*':'*first*'}).

Якщо потрібно обчислити якесь значення за допомогою власної функції (без застосування вбудованих функцій), то для цьогро потрібно використати лямбда-функцію

df.groupby(['*usertype*']).*agg*({'*tripduration*': *lambda x*: *max*(*x*) + 1, '*starttime*':'*first*'}).

1.5 Основні засоби при роботі з декількома таблицями

Різноманітні датасети містять декілька таблиць даних [4, 6]. Для поєднання усіх таблиць, з метою аналізу і виявлення зв'язків між відповідними їх елементами, застосовують метод *join*. Наприклад, для поєднання даних датасетів двох таблиць, у яких є загальне поле (ID об'єкта), можна усі рядки з правої частини таблиці додати їх до лівої частини, або усі рядки з лівої частини додати до правої, або зробити якийсь перетин двох таблиць, або ж взяти якесь їх загальне об'єднання.

У Pandas існує два варіанти поєднання таблиць, а саме: *merge* і *join*. Їх відмінність полягає в тому, що за допомогою методу *merge* пов'язуються дві таблиці за якимось ознаками, а за допомогою *join* пов'язуються дві таблиці за якимось загальним індексом [7].

Наприклад, у нас є датасет із даними оренди квартир в Airbnb у формі трьох таблиць: таблиця *listings* містить детальну інформацію про самі квартири; таблиця *reviews* містить деякі відгуки, які залишали люди; таблиця *calendar* – час і дату бронювання квартири:

```
import pandas as pd
from IPython.display import Image

calendar = pd.read_csv('boston-airbnb-open-data/calendar_sept.csv')
reviews = pd.read_csv('boston-airbnb-open-data/reviews.csv')
listing = pd.read_csv('boston-airbnb-open-data/listing.csv')

calendar.head(2)

reviews.head(2)

listing.head(2).
```

Додамо до таблиці *listings* відгуки людей, використовуючи метод *merge*

pd.merge(*listings*, *reviews*, *left_on*=['*id*'], *right_on*=['*listing_id*']),

результатом якого є певна таблиця.

За замовчуванням метод *merge*, якщо не вказувати, який метод *join* ми хочемо здійснити, здійснює *LEFT JOIN*, тобто додає все значення з правої

таблиці до лівої. Також це можна змінити, просто вказавши в аргументі потрібний метод,

pd.merge(listings, reviews, left_on=['id'], right_on=['listing_id'], how='inner').

Також, якщо потрібно зрозуміти, з якої таблиці і куди додалися значення, для цього потрібно змінити значення у параметра *indicator*, який покаже, з яких таблиць були взяті обрані значення. Тобто, в DataFrame повертається додатковий стовпець *merge*

pd.merge(listings, reviews, left_on=['id'], right_on=['listing_id'], how='inner', indicator=True).

Операція *join* також може бути здійснена при застосуванні іншого методу. Для цього у кожній з таблиць, які поєднуються, задається певний індекс

calendar.set_index('listing_id', inplace=True)

reviews.set_index('listing_id', inplace=True).

Після чого обирається ліва таблиця, наприклад *calendar*, і застосувавши метод *join*, передається таблиця, яка приєднується,

calendar.join(reviews, lsuffix='listing_id', rsuffix='listing_id').

До таблиці *calendar* додалися значення з відгуків, і тепер з'явився новий індекс, що збільшило кількість значень у нашій таблиці.

1.6 Методи перетворення ознак даних

Для розгляду методів перетворення ознак в Pandas завантажимо наш сайт Велобайк по Нью-Йорку, який надає інформацію про поїздки велосипеда (місце виїзду і пункт призначення, тривалість поїздки та тип користувача),

```
import pandas as pd
import numpy as np

df = pd.read_csv('citibike.csv')
df.head().
```

За допомогою функції *map* можна перетворювати елементи стовпця. Для цього створюється словник, де ключу (старе значення стовпця) відповідає нове значення після перетворення.

Змінимо стовпець *usertype*. Оскільки *usertype* містить тільки два поля, тоді *customer*, відповідно, будемо зіставлятись, наприклад із значенням 1, а *subscriber* буде зіставлятись із значенням 2, і тоді з'являється наступний словник

usertype = {'*Customer*':1, '*Subscriber*':2}.

Застосуємо метод *map* до стовпця *usertype*

df['usertype'].map(usertype).head().

Метод *apply* як аргумент передає потрібну нам функцію, яка може бути застосована до всього датасету, або до окремої його колонки.

На прикладі тривалості поїздок стовпець часу, заданий у секундах, перетворюється у хвилини за допомогою лямбда-функції і методу *apply*

df['*tripduration*'].*apply*(*lambda x*: *x* / 60).*head*().

Також можна використовувати власні функції за допомогою методу *apply* і комбінації з лямбда-функцією.

Також *apply* працює з рядками: для цього достатньо вказати *axis* 1 (один із аргументов *apply*). І попередні перетворення можна записати в іншому вигляді

df.apply(*lambda x*: *x*['*tripduration*'] / 60, *axis*=1).*head*().

1.7 Контрольні питання

1. Що таке датасет і які відомі файлові формати їхнього подання ви знаєте?
2. Що таке дистрибутив програмного забезпечення Anaconda? Наведіть відомі аналоги.
3. Призначення бібліотеки NumPy.
4. Призначення бібліотеки SciPy.
5. Призначення бібліотеки Pandas.

2 МЕТОДИ ТА ЗАСОБИ ВІЗУАЛІЗАЦІЇ РЕЗУЛЬТАТІВ АНАЛІЗУ ДАНИХ ПРОГРАМНИМИ ЗАСОБАМИ PYTHON

2.1 Візуалізація даних в бібліотеці Matplotlib

Для реалізації даних в Pandas потрібно створити DataFrame [2, 8], в якому буде дві колонки *x* і *y*:

import pandas as pd

from numpy.random import exponential

df = *pd.DataFrame*({'*x*': *range*(20), '*y*': *exponential*(10,20)}).

Для відображення візуалізованих даних в IPython Notebook потрібно вказати, щоб усе, що рисується в Matplotlib, виводилось усередині, а не в окремому вікні. Для цього потрібно використати команду *magic*

%matplotlib inline.

Наступним береться наш DataFrame і виконується якийсь виклик для візуалізації. Наприклад, виклик *hist*, який відображає гістограму нашого розподілу, де висота стовпчика показує кількість спостережень з таким значенням, яке показано по *x*,

df.y.hist().

Бібліотека в Pandas виконує різні виклики, які дозволяють візуалізувати дані за допомогою гістограми, точок або поділок. Але всі ці способи запозичені з бібліотеки Matplotlib. Для того, щоб почати працювати з бібліо-

текою Matplotlib, доцільно імпортувати її неповністю, а тільки модуль *pyplot*. Модуль *pyplot* дозволяє працювати з бібліотекою Matplotlib в оперативному стилі. Ми вказуємо, що потрібно зобразити, які атрибути змінити, після чого усе це застосується до даного зображення

import matplotlib.pyplot as plt.

На прикладі датасету спортивних результатів гравця зобразимо його успішність залежно від номера спроби. Даний датасет містить один експоненціальний розподіл з параметром $x = 5$, для якого приймається двадцять спроб. І другий експоненціальний розподіл з параметром $x = 6$ і розміром двадцять. Для їх візуалізації потрібно ввести такі команди:

*data*1 = *exponential*(5, 20)

*data*2 = *exponential*(6, 20)

plt.plot(*data*1).

Другий метод зображення даних за допомогою точки з кординатами x і y. Для даного методу є команда *scatter*, в яку необхідно передати два аргументи (x, y)

plt.scatter(*range*(*len*(*data*2)), *data*2).

Було отримано два графіки на одному зображенні. Оскільки вони одного кольору, їх неможливо ідентифікувати. Крім того, відсутні підписи осей для усього графіка. Також відсутня інформація номерів спроб, тому що на осі X поділки зображені не цілими числами, а дробовими. Для усунення вказаних недоліків потрібно налаштувати відповідні графіки.

Для ідентифікації графіка, наприклад для кривої розподілу другого гравця, зарисуємо його червоним кольором

plt.scatter(*range*(*len*(*data*2)), *data*2, *color*='*red*').

Для визначення назви графіка є метод *title*

plt.scatter(*range*(*len*(*data*2)), *data*2, *color*='*red*')

plt.title('*Results*').

Практично будь-якому текстовому об'єкту Matplotlib можна передати параметр *fontdict*, який є словником з деякими атрибутами вашого тексту

plt.title('*Results*', *fontdict*={'*fontsize*': 20}).

Для підпису осей вводиться команди:

plt.xlabel('*Attempt number*')

plt.ylabel('*Result*').

Дані осі також налаштовуються (кут повороту, колір шрифту, розмір тощо). Далі відображення значення графіків певного гравця викликається метод legend, що зображено на графіку

plt.plot(*data*1, *label*='*First player*')

plt.scatter(*range*(*len*(*data*2)), *data*2, *color*='*red*', l*abel*='*Second player*')

plt.title('*Results*', *fontdict*={'*fontsize*': 20})

x.label('*Attempt number*')

y.label('*Result*')

plt.legend().

Легенда автоматично розташовується в місці з мінімальним набором символів даних. Для встановлення заміток по осі *x* у цілочисловій формі потрібно скористатись методом *xtick*

plt.xticks(range(0, 20, 4)).

Для збереження зображення, не тільки в середовищі IPython Notebook [1, 9], але й в інших, потрібно використати метод *savefig*. Важливо вказання правильного розширення файлу, тому що від розширення файлу залежить його формат: якщо файл зберігається як results.png – це буде png-картинка; якщо це буде розширення .pdf – це буде pdf-файл

plt.savefig('*results.pdf*').

2.2 Розширена візуалізація засобаби бібліотеки Matplotlib

Для роботи з бібліотекою Matplotlib [5, 7], з метою визначення способу відображення відразу декількох графіків на одному зображенні, спочатку потрібно задіяти відповідні функції з бібліотек

from numpy.random import exponential

import matplotlib.pyplot as plt

%matplotlib inline.

Для розуміння структури бібліотеки Matplotlib потрібно розглянути базову функцію subplots, яка повертає кортеж з двох об'єктів *fig* і *axes*. При її застосуванні викликається порожній графік з двома осями *x* і *y*

fig, *axes* = *plt.subplots*().

Змінна *figure* містить усі наші зображення, в яких може бути один або більше *axes*. *Axes* – це свого роду полотно, на якому відображаються наші графіки. На одному полотні *axes* можуть будуватись одразу декілька графіків. У полотна *axes* є дві осі – *x* і *y*. Кожному полотну можна задати свій заголовок, свої підписи до осей, свої межі осей тощо.

Для відображення графіка на полотні *axes* потрібно викликати якийсь метод, наприклад, *plot*, але ми викликаємо *plot* не самого модуля *pyplot*, а конкретного об'єкта axes, тим самим визначається, що саме потрібно відобразити безпосередньо на даному полотні

axes.plot(*exponential*(5, 20)).

Для зміни певних властивостей даного полотна використовується команда *set_title*

axes.set_title('*Chart*').

Наша побудова виконується не в якомусь просторі, а на конкретному полотні, властивості якого можуть змінюватися. З даним полотном *axes* можуть виконуватися всі вищенаведені команди (зміна кольору, визначення межі осей тощо). У перспективі з'являється можливість будувати кілька полотен, де кожне з яких має свої координати і параметри налаштування.

Для відображення декілька графіків потрібно вказати кількість рядків у наших графіках і кількість колонок. Для цього так само викликається ме-

тод *subplots* і задається два параметри (*nrows* – кількість рядків, *ncols* – кількість колонок)

fig, axes = plt.subplots(nrows=2, ncols=3).

Для збільшення розміру зображення використовують метод *figsize* (розміри вказуються в дюймах)

fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(10, 5)).

На прикладі заповнення кожного з шести графіків використаємо об'єкт *axes*, який, в даному випадку, це вже не один об'єкт (полотно), а це список списків полотен. І його структура відповідає тому, як виглядає наше зображення. У даному випадку це два списки по три елементи в кожному списку. Відобразимо на кожному полотні свій графік і задамо йому свою назву. Для точної ідентифікації індексів назва буде містити в собі назву колонки і номер рядка. Для цього потрібно проходити за нашим списком списків *axes* і на кожній з осей щось відображати

```
for row, row_axes in enumerate(axes):
    for column, ax in enumerate(row_axes):
        ax.plot(exponential(column, 20))
        ax.set_title('Canvas column {} row {}'.format(column+1, row+1)).
```

Оскільки всі графіки не розміщуються компактно, тоді об'єкту *fig*, який містить усі полотна, потрібно вказати спосіб розміщення з повним вмістом

fig.tight_layout().

Таким чином, можна рухатись по усіх полотнах і на кожному з них виконувати найрізноманітніші операції (відображення різних графіків, із власним набором осей, виконувати власні підписи осей і заголовки). Це дозволяє детально налаштовувати зображення кожного з графіків. За результатами відображення, з метою збереження, потрібно використати об'єкт *fig,* після чого вказуємо *savefig* із місцем його збереження

fig.svefig('all_results.png').

Після чого з'явиться картинка з шістьма графіками.

2.3 Візуалізація результатів аналізу даних засобами бібіліотеки Pandas

Для візуалізації даних, що зберігаються в Pandas Data.Frame, потрібно вміти використовувати бібліотеку Matplotlib [2, 10]. Для цього імпортується безпосередньо сам модуль Matplotlib. Використаємо датасет *titanic.csv*, який зберігає інформацію про пасажирів «Титаніка» (ознаки, факт їхнього виживання в даній катастрофі:

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('titanic.csv')
df.head().
```

Дослідження будь-якого датасету потрібно починати з визначенням властивостей розподілів тих чи інших колонок. Для початку визначимо вартості квитків і яка кількість пасажирів мали дані квитки. Для цього у DataFrame є поле Fare, в якому зберігається інформація про вартість, для якої ми використаємо стандартний метод бібліотеки *pandas* – атрибут *plot*. Атрибут plot є як у колонок, так і в усього DataFrame. Він містить у собі набір методів, які дозволяють викликати ту чи іншу візуалізацію з арсеналу бібліотеки Matplotlib. Наприклад, потрібно розглянути гістограму розподілу. Для цього викликається *hist* і вказується директива *%matplotlib inline* для того, щоб наша візуалізація була відображена всередині нашого Jupyter Notebook [3].

%matplotlib inline

df.Fare.plot.hist().

Отримана гістограма, яка бере наші дані, розподіляє їх по якійсь кількості інтервалів значень і рахує кількість спостережень, які потрапили в той чи інший інтервал. Дана гістограма показує, що найбільше спостережень, або пасажирів, купили найдешевші квитки. Можна деталізувати наші, вказавши кількість колонок, наприклад, двадцять, або зменшити деталізацію, вказавши менше колонок,

df.Fare.plot.hist(*bins*=20)

plt.show().

Можна скористатися іншими методами візуалізації, наприклад, побудувати густину розподілу даного значення. Для цього потрібно викликати у атрибута *plot* метод *kde*:

df.Fare.plot.kde().

plt.show().

Можна будувати розподіл не якоїсь однієї величини, а різних величин. Найпростіше – це *scatter plot*. Для цього потрібно викликати метод вже не безпосередньо в конкретній колонці, а у всьому DataFrame. Відповідно, приймаються якісь два значення *x* і *y*. Для *x* приймається усе той же Fare, а для *y* приймається факт виживання пасажира, атрибут Survived

df.plot.scatter(x='*Fare*', *y*='*Survived*')

plt.show().

Отримано два графіки, які між собою відрізняються, а саме: невизначена належність до Survived, неможливість ідентифікації масштабу, наявність від'ємних значень. Оскільки Matplotlib підтримує два методи роботи з графіками (процедурний [2, 11], об'єктно-орієнтований [3, 12]), то для виправлення даних недоліків можна використати процедурний метод. Для цього імпортується безпосередньо сам модуль Matplotlib

import matplotlib.pyplot as plt.

Після візуалізації *kde* потрібно бачити «легенду»

df.groupby('*Survived*').*Fare.plot.kde*()

plt.legend()

plt.show().

На отриманому графіку від Matplotlib з DataFrame, у нас з'явилася «легенда», що показує: нуль (сині) – це люди, що не вижили; один (помаранчеві) – це люди, що вижили. Також для зміни діапазону значень *x* потрібно вказати *xlim*, щоб він починався як мінімум з нуля, а закінчувався, не враховуючи весь величезний хвіст (200)

```
df.groupby('Survived').Fare.plot.kde()
plt.xlim(0, 200)
plt.show().
```

Для роботи в об'єктному режимі ми можемо викликати будь-який з методів атрибута *plot*, наприклад, усе той же *hist*, але він повертає насправді вже об'єкт *axes*, з яким ми вже працювали. Ми можемо його зберегти в змінну і працювати з нею так, як ми вже працювали з об'єктами Matplotlib. Наприклад, викликається set_title і з'являється заголовок у нашого графіка. Це можна робити і в процедурному стилі, але можна робити і таким чином:

```
ax = df.Fare.plot.hist()
ax.set_title('Visualization').
```

Дане зображення можна також зберегти

```
ax.figure.savefig('something.png')
plt.show().
```

Інформація наявності об'єкта *axes* дозволяє виконувати візуалізації з pandas в інших полотнах, які могли б бути зроблені нами. Для цього потрібно створити наші *figure* і *axes*, як було зроблено раніше, вказавши метод *subplots*,

```
fig, ax = plt.subplots(figsize=(10,5)).
```

Після того, як нами було викликано в атрибута plot метод відтворення чогось, наприклад, *kde*, ми вказуємо на якому з *axes* йому потрібно рисувати

```
df.Survived.plot.kde(label='A11', ax=ax)
for label, class_df in df.groupby('Pclass'):
    print(label)
class_df.Survived.plot.kde(ax=ax, label=label)
plt.legend()
plt.show().
```

На отриманих чотирьох графіках показано: факт виживання людей і врахування їхнього розподілу по класах кабін.

2.4 Інтерактивна візуалізація засобами бібліотеки Plotly

У попередніх підрозділах було розглянуто статичні візуалізації, на основі даних яких виводились графіки та отримувалась картинка.

Для інтерактивної візуалізації даних потрібно використовувати вбудовані елементи бібліотеки візуалізації даних plotly, на основі браузера і JS-бібліотеки [4, 5, 11]. Бібліотека Plotly може бути використана в двох режимах. Один режим – це онлайн-режим, коли у вас використовується сервер plotly. Для цього потрібно отримати логін, пароль, що дозволить зберігати

зображення в хмарі і мати до нього доступ в будь-який момент з будь-якого пристрою

#import plotly.plotly as plt.

Другий спосіб – це оффлайн-режим, коли використовується візуалізація і бібліотека візуалізації, яка локально встановлена на машині. Це дає можливість візуалізувати в браузері з можливістю зберігання в хмарі додатковою кнопкою

import plotly.offline as offline.

Для роботи з *plotly* оффлайн і для створення візуалізації потрібно імпортувати усі графи об'єкта

from plotly.graph_objs import *.

Графи об'єкта – це ті об'єкти, які використовуються, щоб створювати візуалізації. Також для того, щоб візуалізувати дані через plotly або IPython Notebook, потрібно викликати метод *init_notebook_mode*. Даний метод сприймає весь js та всі css-стилі, які потрібні бібліотеці для візуалізації, і вбудовує їх у наш IPython Notebook

offline.init_notebook_mode().

У нього ще є параметр *connected*, який може бути *True* або *False*. За *default* він *False*, і це означає, що використовується JS-бібліотека з plotly package Python, і тому використовувати Інтернет для роботи непотрібно. У другому випадку, якщо *connected* буде *True*, наш IPython Notebook буде мати менший розмір. Більш того? наш IPython Notebook буде використовуватися в реальному часі? для чого з сайту *plotly* завантажиться усі JS-бібліотеки [1].

Оскільки для маніпуляції з нашим dataset буде використовуватись бібліотека Pandas, потрібно взяти dataset, який надає plotly. Цей dataset розкриває дані GDP (рівень валового продукту, тривалість життя в різних країнах і на різних континентах). Для подальшої візуалізації його потрібно підготувати, відсортувавши dataset GDP за валовим продуктом [7, 9]:

import pandas as pd

df = pd.read_csv(https://raw.githubusercontent.com/yankev/test/master/life-expectancy-per-GDP-2007.csv')

df.sort_values('gdp_percap', inplace=True)

df.head().

Для створення візуалізації в *plotly* використовується метод *trace*. *Trace* – це візуалізація якогось одного розподілу даних. У нашому випадку потрібно виконувати *trace* у вигляді *scatter plot*, тобто точок або ліній. Для цього використовується об'єкт scatter

trace = Scatter(x=df.gdp_percap, y=df.lif_exp).

Отже, було передано два параметри. Далі потрібно створити об'єкт data, який інкапсулює в собі усі графіки, які ми хочемо візуалізувати. У даному випадку у нас усього один графік і один *trace*. Тому задається один масив з одного методу *trace*. Далі з цим об'єктом звертаються до plotly (оффлайн версія) і до iplot, повідомляється, а саме: *data*

data = *Data*([*trace*])

offline.iplot(*data*).

Було отримано інтерактивний графік, на якому, якщо навести на конкретне значення, можна побачити конкретні значення *x* і *y*, які приймають наші дані. Також можна виділити частину графіка і побачити більш детальний розподіл у якомусь діапазоні. При натисканні на кнопку зберегти зображення або зберегти його в хмарі, буде запропоновано створити обліковий запис, щоб можна було ділитися нашим зображенням.

Вище було зображено один графік. Для виведення декількох графіків або trace потрібно трохи доопрацювати наш датасет. Виконаємо нижчевказане: введемо нову колонку *population*, куди винесеться з рядка *country* її населення. Введемо колонку *name*, де буде записано назву країни. А також розіб'ємо наш DataFrame на два: один – для континенту Америка, другий – для континенту Європа

df['*population*'] = *df.country.str.split*(':').*apply*(*lambda words*: *float*(*words*[-1]))

df['*name*'] = *df.country.str.split*(':').*apply*(*lambda words*: *words*[1].*split*('<*br*>')[0])

americas = *df*[(*df.continent*=='*Americas*')]

europe = *df*[(*df.continent*=='*Europe*')].

Для отриманих двох DataFrame потрібно створити два *trace*: перший – це той же scatter, але на DataFrame, який належить до *americas*; другий – для *europe*. Обидва утворених *trace* передамо в масив і в об'єкт *data*, щоб відобразити дві візуалізації

*trace*1 = *Scatter*(*x*=*americas.gdp_percap*, *y*=*americas.life_exp*)

*trace*2 = *Scatter*(*x*=*europe.gdp_percap*, y=*europe.life_exp*)

data = *Data*([*trace*1, *trace*2])

offline.iplot(*data*).

Після запуску можна побачимо два *line chart* різного кольору і з легендами. Але легенди не дозволяють нам ідентифікувати відповідні графіки. Для виправлення даної проблеми потрібно модифікувавши наш *trace scatter*. Об'єкт *scatter* і сам *trace* відповідають за те, як виглядає конкретно одна візуалізація. Відповідно, можна як передати дані, так і можна налаштувати якісь параметри, наприклад, налаштування параметра *name*

*trace*1 = *Scatter*(*x*=*americas.gdp_percap*, *y*=*americas.life_exp*, *name*='*Americas*')

*trace*2 = *Scatter*(*x*=*europe.gdp_percap*, *y*=*europe.life_exp*, *name*='*Europe*')

data = *Data*([*trace*1, *trace*2])

offline.iplot(*data*).

Як видно, були видозмінені назви графіків, але потрібно відкорегувати *line chart* шляхом додавання точок, як в *scatter plot*. Для цього налаштовується scatter trace, де вказується його режим, в якому він зображається [3, 7]. У даному випадку вказується режим *markers* для обох графіків і обох *trace*

*trace*1 = *Scatter*(*x=americas.gdp_percap*, *y=americas.life_exp*, *name='Americas'*, *mode='markers'*)

*trace*2 = *Scatter*(*x=europe.gdp_percap*, *y=europe.life_exp*, *name='Europe'*, *mode='markers'*)

data = *Data*([*trace*1, *trace*2])

offline.iplot(*data*).

Було отримано два набори точок. При наведенні на одну з них неможливо ідентифікувати країну, що не дуже зручно. Для того, щоб це виправити, потрібно передати в параметр *text* список з назв наших країн, які також зберігаються в нашому DataFrame,

```
trace1 = Scatter(
    x=americas.gdp_percap,
    y=americas.life_exp,
    name='Americas',
    mode='markers',
    text=americas.name
)
trace2 = Scatter(
    x=europe.gdp_percap,
    y=europe.life_exp,
    name='Europe', mode='markers',
    text=europe.name
)

data = Data([trace1, trace2])

offline.iplot(data).
```

Для надання більшого контексту нашій візуалізації можна змінювати розмір точки залежно від якогось параметра, наприклад, залежно від розміру населення. Дане поле зберігається в DataFrame, в ключі *population*, де параметр *marker* відповідає за розмір точки [2, 10].

Для налаштування *marker* потрібно вказати, що *marker* – це словник, у якого є ключ *size* аналогічний, як і в *population*. Але якщо вказати просто population, то розміри будуть мільйонними, а це заповнить нам увесь графік. Тому нам потрібно нормалізувати це на максимальне значення населення, яке множимо, наприклад на 20, щоб отримати розмір наших точок від нуля до 20

```
trace1 = Scatter(
    x=americas.gdp_percap,
    y=americas.life_exp,
    name='Americas', mode='markers',
    text=americas.name,
    marker={'size': americas.population/americas.population.max()*20}
)
trace2 = Scatter(
    x=europe.gdp_percap, y=europe.life_exp,
    name='Europe', mode='markers',
    text=europe.name
)
data = Data([trace1, trace2])
offline.iplot(data).
```

Отримані точки різного розміру дозволяють виконувати наприклад, аналіз очікуваної тривалості життя (велика країна з високим GDP має очікувану високу тривалість життя). Для досить малих країн, яких неможливо на графіку побачити, можливості бібліотеки plotly дозволяють шляхом масштабування також їх детально проаналізувати. Крім того, особливості бібліотеки plotly дозволяють побачити більший діапазон значень, порівняно зі статичною візуалізацією. Але, в той же час, характеристика розміру країни за допомогою розміру точки – не дуже гарне рішення, тому що багато точок (малі країни) після цього зникли.

2.5 Контрольні питання

1. На основі яких системних додатків працюють вбудовані елементи бібліотеки візуалізації даних plotly?
2. Яким чином виконується збереження результатів аналізу даних засобами бібліотек Matplotlib і Plotly?
3. Призначення бібліотеки Matplotlib.
4. Призначення бібліотеки Plotly.

3 МЕТОДИ ТА ЗАСОБИ СТАТИСТИЧНОГО АНАЛІЗУ ДАНИХ

3.1 Засоби реалізації статистичних методів в бібліотеці SciPy

Для обробки статистичних даних будуть використовуватись бібліотеки NumPy і SciPy, а точніше – модуль stats [1, 3, 5]

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
```

%matplotlib inline

plt.style.use('ggplot').

Для початку потрібно, щоб були розглянуті принципи створення дискретного і неперервного розподілу вже відомого типу. Але перш ніж переходити безпосередньо до них, потрібно розібратися як у самому модулі реалізовані класи, що відповідають за випадкові величини.

У *stats* є два класи – *rv_continuous* і *rv_discrete*, в яких реалізовані всі методи для роботи з випадковими величинами. Можна порахувати функцію розподілу, підрахувати густину ймовірності для неперервної випадкової величини, різні статистики тобто, всі потрібні методи вже реалізовані в даному класі. Для створення розподілу відомого типу потрібно, щоб був створений клас, який успадковується від цих двох класів,

stats. rv_continuous

stats.rv_discrete.

Якщо потрібно створювати власні випадкові величини, то можна успадковувати від цих класів та їх використовувати.

Для роботи з дискретними розподілами доцільно розпочати з розподілу Бернуллі (випадкова величина приймає тільки два значення: нуль або одиниця з певною ймовірністю) [8]. Для створення випадкової величини потрібно вказати відповідний клас, який називається так само, як і випадкова величина, і задати йому єдиний параметр

rv_bernoulli = stats.bernoulli(*p*=0.3).

Після чого ініціалізується об'єкт, на основі якого створюються вибірки. Для цього достатньо, щоб були задані тільки розмір вибірки і отримано масив

rv_bernoulli.rvs(14).

plt.hist(*rv_bernoulli.rvs*(14), *bins*=10)

plt.show().

При біноміальному розподілі (узагальнення розподілу Бернуллі) [11] розглядається не один експеримент, а *n* експериментів з ймовірністю успіху *p*. Тоді для створення об'єкта, який відповідає за біномінальний розподіл, потрібно вказати два параметри: кількість експериментів і ймовірність

rv_binom = stats.binom(100, *p*=0.9).

На основі даного розподілу може бути згенерована деяка вибірка

rv_binom.rvs(8).

Для відображення гістограми потрібно ввести

plt.hist(*rv_binom.rvs*(80), *bins*=10)

plt.show().

По осі *x* відкладаються значення з нашої вибірки, а по осі *y* відкладається та кількість разів, скільки дане значення зустрілося у вибірці.

Переходячи до неперервного розподілу [13] доцільно розпочати з рівномірного (випадкова величина визначена на деякому відрізку, де зустрічається з однаковою ймовірністю). За межами даного відрізка ймовірність зустріти випадкову величину дорівнює нулю.

Для того, щоб створити випадкову величину рівномірного розподілу, потрібно вказати початок відрізка, а також на скільки одиниць потрібно зміститись від стартової точки. Тобто, задається не початок і кінець, а саме старт і зміщення

$a = 5$

$b = 10$

rv_uniform = stats.uniform(a, b - a).

Після того, як було створено об'єкт випадкової величини, може бути визначена функція розподілу в певній точці. Якщо приймається значення з нашого інтервалу, то функція розподілу не дорівнюватиме нулю. Якщо приймаються значення, що не потрапляють в наш інтервал, тоді значення функції розподілу дорівнюватиме або одиниці, або нулю

rv_uniform.cdf(5.5).

Аналогічно і з густиною ймовірності, тобто, для значень усередині інтервалу густина ймовірності не дорівнює нулю, а для значень поза інтервалу густина ймовірності буде дорівнювати нулю

rv_uniform.pdf(7).

Для отримання графіка функції розподілу потрібно створити певні значення, які передаються в нашу функцію розподілу, в результаті чого буде отримано масив значень функції розподілу

X = np.linspace(a - 2, b + 2, 100)

cdf = rv_uniform.cdf(X)

plt.plot(X, cdf)

plt.ylabel('F(x)')

plt.xlabel('x')

plt.ylim([0,1.5])

plt.title('Cumulative distribution function for uniform')

plt.show().

За результатами виведення видно, що наша випадкова величина визначена на відрізку [5, 10], а за його межами значення буде дорівнювати нулю або одиниці. Перейдемо до побудови графіка густини ймовірності. Оскільки густина ймовірності – це похідна від функції розподілу, то на певному відрізку густина ймовірності величина постійна

X = np.linspace(a-2 , b + 2, 1000)

pdf = rv_uniform.pdf(X)

plt.plot(X, pdf)

plt.ylabel('f(x)')

plt.xlabel('x')

plt.xlim([4, 12])

plt.title('PDF for uniform').

При роботі з нормальним розподілом для його створення потрібно задати два параметри: μ – середнє значення і σ – середньоквадратичне відхилення. Для створення об'єкта нормальної величини потрібно, щоб були визначені такі аргументи, як *loc*, що відповідає за середнє значення і *scale* – це буде наша сигма

```
mu = 2
sigma = 0.5

rv_norm = stats.norm(loc=mu, scale=sigma).
```

Також з цієї випадкової величини можна зробити sample даних

```
rv_norm.rvs(17).
```

Графік функції розподілу для нормальної величини

```
x = np.linspace(0, 4, 100)
cdf = rv_norm.cdf(x)
plt.plot(x, cdf)
plt.ylabel('F(x)')
plt.xlabel('x')
plt.show().
```

У даному випадку характер функції розподілу не змінюється (функція визначена від нуля до одиниці і не убуває), але графік вже більш згладжений.

Побудуємо графік густини ймовірності

```
x = np.linspace(0,4,100)
pdf = rv_norm.pdf(x)

plt.plot(x, pdf)
plt.ylabel('f(x)')
plt.xlabel('x').
```

Оскільки нормальний розподіл симетричний, то і графік також симетричний. Самостійно змініть параметри нормального розподілу, наприклад, фіксуючи μ і змінюючи середньоквадратичне відхилення. Подивіться, як густина ймовірності буде змінюватися. Наприклад, якщо значення σ мале, то і графік більш вузький, чим більше значення σ, тим графік виглядає більш широким

```
means = [1,5,10]
mu = 10
sigmas = [1, 2, 4, 6]
for sigma in sigmas:
    rv_norm = stats.norm(loc=mu, scale=sigma)
    x = np.linspace(0,20,200)
    cdf = rv_norm.pdf(x)
    plt.plot(x, cdf, label=sigma)
plt.legend().
```

Крім того, може бути обчислена функція розподілу густини ймовірності, використовуючи бібліотеку *stats* для обчислення різних статистик. Для цього для нашої величини може бути викликаний метод *stats* і обрано відповідні моменти. Тобто, може бути пораховано середнє значення *mean*, дисперсія *var* і параметр зміщення *skew* (тобто, якщо розподіл асиметричний, наприклад зміщення вліво або вправо, тоді даний параметр не дорівнюватиме нулю)

```
from scipy import mean, var
from scipy.stats import skew

mean, var, skew = rv_norm.stats(moments='mvs')
print(mean, var, skew).
```

3.2 Довірчий інтервал в задачах інтелектуального аналізу даних

Довірчий інтервал – це показник точності вимірювань. Його застосовують як для отримання оцінки середніх значень, так і для отримання оцінки дисперсії. Він також відображає, наскільки величина, що визначена за вибіркою, відображає справжнє значення за деякою генеральною сукупністю [4, 12].

Загалом довірчий інтервал визначається через ймовірність того, що оцінюваний параметр (це може бути середнє значення або дисперсія) не виходить за певні межі (певний рівень довіри) [2, 9]. Тобто, це ймовірність того, що довірчий інтервал містить точні значення. Зазвичай приймається рівень довіри, який дорівнює 0,95 або 0,99.

$P(LB \le \theta \le RB) = p$, де p – рівень довіри, а ліву LB і праву RB межі потрібно знайти.

Для нормального розподілу, у випадку відомої дисперсії, формула довірчого інтервалу виглядає так:

$$P\left(\tilde{X} - z_{1-\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}} \le \mu \le \tilde{X} + z_{1-\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}}\right) = 1-\alpha, \qquad (3.1)$$

де $\tilde{X}$ – виборче середнє, σ – відома дисперсія генеральної сукупності, тобто по усьому розподілу, n – розмір вибірки.

```
from PIL import Image
Im=Image.open('The_Normal_Distribution.png')
im.show().
```

Тобто, з (3.1) потрібно визначити межі, в яких знаходиться якесь середнє значення. У (3.1) невідомий параметр z визначається з правила двох і трьох сигм (95% усіх значень містяться в межах $\pm 2\sigma$, а 99% значень знаходиться в інтервалі $\pm 3\sigma$) за допомогою функції розподілу, яка насправді накопичує усі значення. Тобто, якщо потрібно вибрати рівень довіри, який

дорівнює 0,95, то відповідне значення функції розподілу буде близько до 97% [4, 9]. Після чого обчислюється деякий *z-score*, ґрунтуючись на функції розподілу, а ці значення, у свою чергу, будуть задавати межі інтервалу.

Для обчислення довірчого інтервалу, використовуючи бібліотеку SciPy, зафіксовується деякий вихідний розподіл з певними параметрами

from scipy import stats

population = stats.norm.rvs(loc=2, scale=5, size=100000).

Після чого потрібно зробити *sample* даного розподілу. Для цього можна скористатися методом *np.random.choice*, де передається вихідний розподіл і вказується, який розмір вибірки потрібно отримати

import numpy as np

sample_size = 100

sample = np.random.choice(a = population, size = sample_size).

Визначається вибіркове середнє і дисперсія генеральної сукупності

sample_mean = sample.mean()

st_dev = population.std().

Після чого визначається відповідне *z-value*

z_value = stats.norm.ppf(q = 0.975)

print("z-value:right", z_value).

Потрібно зауважити, що нормальний розподіл симетричний, і тому достатньо знайти *z-value* з одного кінця, тому що в силу симетричності ліва межа буде дорівнювати правій, але з протилежним знаком

z_value = stats.norm.ppf(q = 0.025)

print("z-value:left", z_value).

Тут використовується функція, яка називається *probability percentile function* (*ppf*) – це функція, обернена функції розподілу, в якій просто задається відсоток і обчислюється *z-value*. Тоді загалом довірчий інтервал визначається так:

*interval = z_value * (st_dev/np.sqrt(sample_size))*

conf_inv = (sample_mean - interval, sample_mean + interval)

print("Confidence interval:", conf_inv).

Усі отримані вище обчислення можна зібрати в одну функцію і можуть бути застосовані для іншої вибірки

```
def compute_ci(sample, st_dev):

    z_value = stats.norm.ppf(q = 0.975)
    sample_size = len(sample)
    interval = z_value * (st_dev/np.sqrt(sample_size))
    conf_inv = (sample_mean - interval, sample_mean + interval)

    return conf_inv.
```

Для прикладу, може бути обрана вибірка більшого розміру, ніж вихідна

```
np.random.seed(5)
sample_size = 2000
sample = np.random.choice(a = population, size = sample_size)

ci = compute_ci(sample, st_dev)

print("conf interval for 2000 sample size:", ci).
```

Потрібно зауважити, що в цьому випадку довірчий інтервал став вузьким. Тобто чим більше вибірка, тим точніше ми можемо оцінити параметри розподілу.

Вище було розглянуто випадки з відомою дисперсією. Але якщо задана дуже маленька вибірка і не можна оцінити за нею дисперсію, на допомогу приходить розподіл Стьюдента. Потрібно розглянути як у даному випадку буде виглядати формула довірчого інтервалу. Також потрібно оцінити середнє [4, 5, 13]

$$P\left(\tilde{X} - z_{1-\frac{\alpha}{2},n-1}\frac{S_0}{\sqrt{n}} \le \mu \le \tilde{X} + z_{1-\frac{\alpha}{2},n-1}\frac{S_0}{\sqrt{n}}\right) = 1-\alpha,$$

де $\tilde{X}$ – виборче середнє, S_0 – виправлена виборча дисперсія, n – розмір вибірки.

Для визначення у даному випадку довірчого інтервалу потрібно скласти деяку функцію. Обчислення розподілу Стьюдента вимагає використання відповідного класу t і з нього викликається той метод, який викликається в разі нормального розподілу. Після чого передаються параметри, які будуть відповідати нашому інтервалу

```
def compute_ci_t(sample, alpha=0.95):

    n = sample.shape[0]
    mu, se = np.mean(sample), stats.sem(sample)
    bound = se * stats.t.ppf((1 + alpha) / 2., n-1)

    return mu - bound, mu + bound
sample = np.random.choice(a = population, size = 30)
ci_t = compute_ci_t(sample, alpha=0.95)
print("conf interval with t test for 2000 sample size:", ci_t).
```

Аналізуючи довірчий інтервал потрібно зауважити, що в цьому випадку довірчий інтервал став суттєво більше, тому що дисперсія невідома. Тому є сенс його розширювати.

Було розглянуто, як обчислюється довірчий інтервал для нормального розподілу. Але найчастіше на практиці є деяка вибірка, і невідомо, якого вона походження. Як в цьому випадку, може бути оцінена якість параметрів

цієї вибірки і як зрозуміти, що дійсно вибірка репрезентативна і статистично значуща? У цьому випадку має місце центральна гранична теорема (ЦГТ) [1, 4]. Відомо, що розподіл середніх – це і є нормальний розподіл. А для нормального розподілу вже відомо, як обчислюється довірчий інтервал.

Довірчий інтервал на прикладі поїздок таксі в місто Мехіко визначається як вказано нижче.

```
import pandas as pd
import numpy as np
taxi_mex = pd.read_csv('taxi-route/mex_clean.csv')

def generate_distribution_sample(data, sample_size, dist_size):

    sample_means = []
    for i in range(dist_size):
        sample = np.random.choice(a = data, size = sample_size)
        sample_means.append(np.mean(sample))

    return sample_means.
```

Для початку потрібно згенерувати розподіл середніх. Для цього потрібно задати розмір вибірки і згенерувати таких вибірок дуже багато. Адже, як відомо, однією з умов теореми є те, що чим більше розмір вибірки, тим ближче розподіл середніх до нормального розподілу

```
sample_size = 10000
dist_size = 50000
sample_means = generate_distribution_sample(taxi_mex['dist_meters']/1000,
sample_size, dist_size).
```

Давайте тепер подивимося, як виглядає гістограма для розподілу середніх

```
import matplotlib.pyplot as plt
plt.hist(sample_means, bins=100)
plt.xlabel('distance in km')
plt.show().
```

Потрібно відмітити, що воно вже більш симетричне і можна зрозуміти, який розкид значень у цього параметра. Оскільки це вже нормальний розподіл, то можна обчислити відповідні процентні співвідношення, і після цього отримати довірчий інтервал. Для тестування можна взяти відстань в кілометрах, яку проїжджає таксі, і побудувати для цього значення довірчий інтервал.

```
np.sort(sample_means)
lb = np.percentile(sample_means, 2.5)
ub = np.percentile(sample_means, 97.5)
print("conf interval for bootstrap:", (lb, ub)).
```

3.3 Перевірка гіпотез і розподіл Стьюдента

У статистиці *t*-критерій Стьюдента – це набір методів для перевірки гіпотез [2, 6, 7], які часто його використовують для перевірки рівності середніх значень вибірки. Для цього потрібно, щоб було обчислена спочатку *t*-статистика, після чого її потрібно порівняти з деяким граничним значенням, що дозволить прийняти рішення про прийняття або відхилення самої гіпотези.

T-статистика будується за таким принципом: у чисельнику вказується величина з нульовим математичним очікуванням, а в знаменнику – стандартне відхилення від цієї випадкової величини.

Алгоритм перевірки гіпотез побудований за таким принципом: для початку потрібно, щоб була визначена деяка нульова гіпотеза. У разі порівняння середніх потрібно прийняти припущення, що середні значення за двома вибірками рівні. Також визначається альтернативне їй припущення – нерівність середніх значень. Після цього обчислюється *t*-статистика і порівнюється з показником *p-value* (ймовірність прийняття нульової гіпотези).

P-value – це ймовірність прийняття нульової гіпотези за умови, що справедлива альтернативна гіпотеза. Тобто, якщо *p-value* дуже маленьке значення, то, швидше за все, не відбуваються взагалі якісь випадкові процеси. Але якщо ж *p-value* досить велике, то ймовірність того, що вибірки, отримані випадковим чином, не мають ніяких зв'язків, дуже значна.

Розглянемо три випадки застосування критерію Стьюдента:

- перший випадок – порівнюється середнє значення за вибіркою з якимось відомим середнім значенням;
- другий випадок – порівнюються два значення за двома незалежними вибірками;
- третій випадок – у даній серії вимірювань перевіряється наскільки статистично значимі усі виміри.

Для переходу до одновибірного критерію, на прикладі dataset поїздок по містах Мехіко і Богота, потрібно перевірити чи відрізняється тривалість поїздок у цих містах залежно від часу очікування таксі

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
taxi_bog = pd.read_csv('taxi-route/bog_clean.csv')
taxi_mex = pd.read_csv('taxi-route/mex_clean.csv').
```

Наприклад, для перевірки часу очікування таксі потрібно припустити, що середній час очікування – це десять хвилин. Для цього в *stats* реалізовані усі методи і тому для перевірки одновибірного критерію потрібно використати метод *ttest_1samp*. При цьому потрібно передати нашу вибірку і також передати те значення середнього, яке очікується побачити.

Результатом роботи даного методу буде певний клас, який повертає два параметри: *statistic* – це і є *t*-статистика, а також розрахований *pvalue*

sample = taxi_mex['wait_sec'].sample(n=3000)/60

print(stats.ttest_1samp(sample, 10)).

Для того, щоб зрозуміти, чи можна відкинути або прийняти нульову гіпотезу, потрібно проаналізувати значення *p-value*. Зазвичай *p-value* приймається рівним 0,05 або 0,01. Але тут фіксується значення, рівне 0,5. Потрібно відмітити, що для цієї гіпотези *p-value* менше, ніж 0,5; тобто, виходить, що нерівні середні, значить, відкидається нульова гіпотеза.

Аналізуються дві вибірки, а саме: бази даних поїздок по місту Мехіко порівнюються з поїздками по Боготі.

taxi_mex['pickup_datetime'] = pd.to_datetime(taxi_mex.pickup_datetime)

taxi_mex['month'] = taxi_mex['pickup_datetime'].dt.month

taxi_bog.shape

taxi_mex.shape.

При цьому припускається, що середня тривалість поїздок у двох містах однакова. Також може бути підрахована й *t*-статистика, але для цього потрібно використати інший метод, який називається ttest_ind, куди передаються вже наші обидві вибірки. Після цього повертається так само, як і у попередньому випадку, обчислена статистика *p-value*.

print(stats.ttest_ind(taxi_mex['trip_duration'].sample(n=3000),

taxi_bog['trip_duration'].sample(n=3000))).

У даному випадку значення *p-value* дуже близьке до «0», тоді нами знову відкидається нульова гіпотеза [4, 5]. Тобто, тривалість поїздок відрізняється. Але у разі врахування часу очікування таксі, нами вже не може бути відкинута нульова гіпотеза, що, в принципі, логічно (приблизно в середньому люди очікують таксі однаковий час).

print(stats.ttest_ind(taxi_mex['wait_sec'].sample(n=3000),

taxi_bog['wait_sec'].sample(n=3000))).

Необхідно визначити, чим відрізняється тривалість поїздок в одному місті, але в різні місяці, на прикладі поїздок по місту Мехіко за листопад і грудень. У даному випадку використовується також метод з SciPy, який називається *ttest_rel*. У даному випадку передаються вже дві вибірки.

control = taxi_mex[taxi_mex.month ==11]['trip_duration'].sample(n=1000)

treatment = taxi_mex[taxi_mex.month ==

12]['trip_duration'].sample(n=1000)

print(stats.*ttest_rel*(*control*, *treatment*)).

У даному випадку *p-value* набагато більша, ніж рівень значимості. Тобто, вже не можна відкинути нульову гіпотезу, що, в принципі, і логічно. Можна припустити, що листопад і грудень так близько знаходяться в календарі, що, швидше за все, погодні умови ніяк не впливають на тривалість поїздок.

3.4 Контрольні питання

1. Що таке розподіл Стьюдента і довірчий інтервал?
2. У чому відмінність між біноміальним і нормальним розподілом? Які їх призначення і характеристики?
3. Що таке центральна гранична теорема і яке її призначення?
4. Наведіть алгоритм перевірки статистичних гіпотез?
5. Що таке *t*-критерій Стьюдента?

4 ЛІНІЙНІ МОДЕЛІ В ЗАДАЧАХ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ

4.1 Види методів машинного навчання

Найпоширенішим підходом до машинного навчання є навчання «з учителем» або «super viselearning» [9, 11, 13]. У даному випадку є якась кількість розмічених даних, для яких визначена коректна відповідь і коректне передбачення. На цих даних виконується навчання нашої моделі, після чого виконується передбачення на нових даних, з якими ще не були знайомі.

У даному випадку навчання виконується з учителем, і класичним прикладом задачі, яка була вирішена і вирішується на даний момент за допомогою даного підходу і машинного навчання, є задача кредитної оцінку [1, 8]. Відповідно до цієї задачі існує якийсь банк і пул клієнтів, для яких є інформація (рівень власних прибутків, а також факт отримання і повернення кредитів). Наприклад, якщо потрібно визначити, чи варто видати кредитні кошти певному клієнту банка і на яку суму, на основі даних отриманих із заповненої інформаційної анкети, тоді це задача навчання з учителем (на основі нових даних виконується передбачення). Якщо потрібно тільки визначити, чи варто видавати кредит чи ні, тоді це є завдання класифікації. Якщо заодно потрібно визначити, на яку суму видати кредитні кошти, тоді це називається завдання регресії (відповідь передбачення є дійсним числом). Оскільки у даний час вищевказаний підхід найбільш поширений; якщо певні компанії або підприємства, що застосовують машинне навчання в своїй роботі, то, швидше за все, вони використовують підхід навчання з учителем. Відповідні методи і моделі є найефективнішими через найкраще навчання, тому й застосовуються найчастіше.

Велика частина нейронних мереж працює на певних розмічених даних. Але якщо навчання відбувається на нерозмічених даних і для них не потрібна якась ручна або навіть автоматична розмітка, то даний підхід навчання називається «без учителя», або «unsupervised learning».

У даному випадку використовується певна залежність всередині даних і робляться певні висновки за нашими даними або якісь перетворення. На-

приклад, класичною задачею для навчання без учителя є задача зниження розмірності. Наприклад, для певного датасету, набору даних з великою кількістю ознак, потрібно: щоб навчання проводилось не на величезній кількості ознак, наприклад, не на тисячі ознак, а тільки на ста, ста найважливіших; також потрібно відкинути зайві ознаки, зберігаючи максимальну предикативну здатність, тобто так, щоб згодом була можливість використовувати дані ознаки в роботі. Таким чином здійснюється зниження розмірності даних, відкидаються ознаки, які найменш впливають на зміст та сутність цих даних. Також є можливість не просто відкидати ознаки, але й приходити в якийсь новий простір ознак, генеруючи нові або просто моделюючи якийсь новий простір.

Ще одним із завдань даного підходу є задача кластеризації, де у великій кількості даних визначаються певні кластери або групи схожих один на одного об'єктів. Причому у даному випадку також відсутня розмітка, тому що виконується певне групування вихідних даних на певну кількість кластерів.

Останнім часом значного поширення отримала задача моделювання статистичного розподілу даних, де будується модель, яка здатна моделювати розподіл цих даних з метою генерації: нових прикладів, нових ознак і нових даних.

Однією з найбільш поширених застосувань підходу навчання «без учителя» в різних галузях людської діяльності є задача пошуку аномалій. Наприклад, у певному банку здійснюється велика кількість транзакцій, і потрібно проаналізувати дані транзакції з метою визначення шахрайських транзакцій. У даному випадку, як правило, мала кількість прикладів аномальних транзакцій, шахрайств, а, можливо, вони й взагалі відстуні. Відбувається пошук якихось розкидів, аномальних транзакцій, які можуть бути «фродом», або шахрайством. Аналогічно можна застосовувати для певних заводів і складних пристроїв, де поломки трапляються дуже рідко або навіть ніколи.

Ще один метод, який дещо відрізняється від попередніх, – це навчання «з підкріпленням» або «reinforcement learning». У даному випадку постановка задачі відбувається трохи по-іншому. Існує певне середовище, або оточення, і агент, який діє у цьому середовищі. Тобто, якісь задані правила в нашому світі, якась фізика в цьому світі, і якийсь агент (актор), що намагається з цим світом взаємодіяти, отримуючи винагороду або штрафи за свої дії. Таким чином, повторюючи різні дії, наш агент може вивчати певних політиків, вивчати певну поведінку, щоб максимізувати, наприклад, винагороду. Це дуже потужна концепція, яка використовується на даний момент досить обмежено, проте має велику перспективу використання в майбутньому [5, 7].

На даний момент більшість наукових досліджень з даної тематики працює з деякими ігровими середовищами (AlphaGo від компанії DeepMind, який не так давно обіграв чемпіонів світу з гри в Go [1, 3]).

Але, незважаючи на те, що дуже багато досліджень зосереджено саме на ігрових середовищах, також їм знаходиться застосування в реальному житті [2, 5]. Наприклад, за допомогою підходу навчання «з підкріпленням» була оптимізована робота для центру Google, що дозволило зекономити споживання електроенергії практично вдвічі [4, 9].

Також активно підхід навчання «з підкріпленням» використовується в робототехніці, тому що задача, коли агент діє в оточенні і отримує винагороду, відмінно відображається на історії (наприклад, коли робот вчиться ходити або літати, брати якісь предмети, щось в загальному розпізнавати, взаємодіяти з реальним світом). У даному випадку можна не тільки навчити робота в реальному світі, але й перенавчати його в якомусь віртуальному середовищі, що дозволить йому згодом взаємодіяти з навколишнім світом і донавчатися.

Також навчання «з підкріпленням» активно використовується в трейдингу [2], коли розроблені боти [5] можуть здійснювати операції купівлі або продажу. І для даних операцій існує чітка винагорода – це прибуток, який був отриманий або не отриманий у результаті купівлі або продажу якихось активів. За допомогою підходу навчання «з підкріпленням» можна навчити відповідних ботів, які діють на ринках.

Також існують проміжні підходи між вищевказаними підходами машинного навчання, а саме «semi-supervised learning». Даний підхід може не просто використовувати розмічені дані, але й, на додаток до розмічених даних, може використовувати певну кількість нерозмічених даних.

4.2 Лінійна регресія

Лінійна регресія вирішує задачу підходу «навчання з учителем». Це означає, що є певний розмічений набір даних, на яких виконується навчання. Після чого відбувається передбачення на нових даних, які ще невідомі. «Лінійна регресія» передбачає якусь дійсну змінну, що дозволяє здійснювати моделювання лінійної залежності від заданих ознак. Тобто, існує певна змінна, яка залежить, як очікується, від інших ознак і для якої виконується пошук залежності.

Наприклад, можна спробувати передбачити заробітну плату фахівця залежно від його віку, досвіду, статі тощо. Можна очікувати, що заробітна плата фахівця залежить від віку і досвіду, але не залежить, наприклад, від статі. Цю залежність і потрібно змоделювати.

Наприклад, для задачі парної регресії (побудова залежності однієї змінної від іншої), потрібно побудувати залежність ваги людини від її зросту. Очевидно, існує певна природна залежність, яку можна побудувати. Тобто, існує функціонально-залежна пряма в двовимірному просторі, яка може бути використана для передбачення ваги будь-якої іншої людини.

Проте в реальному світі простір даних є простором набагато більшої розмірності (площини або гіперплощини).

Таким чином, лінійна регресія – це цільова функція, для якої потрібно визначити різні вагові коефіцієнти (характеристика різних ознак), які найкраще описують вихідні дані.

$$Y = \omega_0 + \omega_1 x_1 + \ldots + \omega_k x_k = \langle \omega, x \rangle, \tag{4.1}$$

де Y – цільова змінна, x_k – ознаки, ω_k – вагові коефіцієнти.

Також цільова функція лінійної регресії може бути інтерпретована як зважена сума різних ознак моделі у вигляді вагових коефіцієнтів, що будуть підбиратися в нашій моделі. За значеннями підібраних вагових коефіцієнтів визначається, які з ознак є найбільш важливими.

Узагальненням лінійної регресії є поліноміальна регресія, яка виконує аналогічні функції. У даній цільовій функції аналогічно підбираються вагові коефіцієнти характеристик ознак моделі, проте змінні характеристики ознак можуть бути піднесені у степеневу функцію.

4.3 Функціонал якості і градієнтний спуск

При роботі з лінійною регресією використовується підхід навчання «з учителем», де на розмічених даних виконується процес навчання моделі з метою передбачення на нових даних. Для визначення характеристики якості опису даних тренувальної вибірки, на якій відбувається процес навчання, потрібно функція помилки, яку необхідно оптимізувати.

Як функція помилки може бути використана «Mean Absolute Error», або середня абсолютна помилка, яка вказує, як далеко середні прогнози лежать від коректних відповідей:

$$MAE = \frac{1}{N} \sum_{i=1}^{N} \left| y_i - \hat{y}_i \right|, \tag{4.2}$$

де N – розмір вибірки, y_i – коректна відповідь, $\hat{y}_i$ – передбачення.

Залежність (4.2) достатньо логічна і добре інтерпретована функція втрат, проте вона не диференційовна, тому не може бути використана, наприклад, для методу градієнтного спуску [5] або методів градієнтної оптимізації [2, 6] взагалі. Тому, зазвичай, використовуються інші функції. Наприклад, функція середньої квадратичної помилки («Mean Squared Error»), з аналогічною функцією, але з усередненням квадратів відстаней передбачень від реальних відповідей:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}_i \right)^2, \tag{4.3}$$

де N – розмір вибірки, y_i – коректна відповідь, $\hat{y}_i$ – передбачення.

Дана функція (4.3) диференційовна і дуже часто використовується в методах градієнтного спуску. Однак лінійна регресія з такою функцією втрат може вирішувати проблему й аналітично, без використання градієнтного спуску. Але в просторах великої розмірності доводиться оперувати складними матрицями, й, відповідно, не завжди це працює. Тому дуже часто легше використовувати градієнтні методи, які дають результат практично завжди.

Отже, функція втрат може бути подана поверхнею, по якій потрібно спуститися. Тобто, потрібно мінімізувати функцію, визначити її мінімум. Саме там модель і буде працювати найкраще. Загальновідомо, що градієнт показує напрямок найшвидшого зростання функції, а антиградієнт – напрямок найшвидшого зменшення функції. Тому можна взяти якийсь набір параметрів (вектор) у нашій моделі, визначити вектор антиградієнта і по ньому спуститися. Використовуючи вектор антиградієнта параметри моделі оновлюються до тих пір, поки відбудеться збіжність, або якість нашої моделі не буде влаштовувати задані характеристики. Тобто, послідовно оновлюються параметри моделі, наприклад, вагові коефіцієнти лінійної регресії відповідно до вектора антиградієнта.

Існує величезна кількість модифікацій градієнтного спуску: можна змінювати швидкість ітераційного кроку, можна оперувати підвибірками об'єктів. Наприклад, існує стохастичний градієнтний спуск [10], який використовує не всю вибірку для ітераційних кроків за антиградієнтом, а тільки один об'єкт, або градієнтний спуск «mini-batch» (набір об'єктів). Усі види модифікацій і методи оптимізації, які використовуються як в лінійній регресії, так і, наприклад, у нейромережах, зазвичай, це певні різновиди градієнтного спуску.

4.4 Логістична регресія

У задачах класифікації передбачається якась дискретна відповідь, для чого використовується підхід машинного навчання «з учителем». Тобто, процес навчання відбувається на різних даних, виконується прогноз нових даних і передбачається дискретна відповідь (0 або 1 в разі бінарної класифікації).

Наприклад, можна класифікувати спам або виконувати більш складну багатокласову класифікацію (тварини, рослини тощо). У разі бінарної класифікації можна використовувати лінійну регресію і дивитися на знак отриманого числа. Лінійна регресія передбачає дійсне число, і, наприклад, якщо у нас значення менше нуля, тоді вважається, що це нуль, а якщо більше нуля, тоді це вважається одиниця. Таким чином, для лінійно роздільних класів виконується побудова деякої роздільної поверхні, де усі точки, що лежать зверху, будуть відноситись до класу «1», а усі точки, що лежать знизу, будуть відноситись до класу «0».

Але, якщо необхідно передбачати ймовірність належності до якогось класу, тоді потрібно використовувати логістичну регресію. Логістична регресія в своїй основі використовує сигмоїд-функцію, в яку відправляється лінійний класифікатор, а сигмоїд-функція, у свою чергу, повертає число від «0» до «1», яке може бути віднесене відповідно до деякої ймовірності:

$$z = w_0 + w_1 x_1 + \ldots + w_k x_k, \tag{4.4}$$

$$y = sigmoid(z) = 1/\left(1 + e^{-z}\right). \tag{4.5}$$

Чим ближче число до одиниці, тим вища ймовірність належності до певного класу. Також можна побудувати деякий класифікатор, вибираючи певне відсікання «decision boundary».

Для оптимізації логістичної регресії використовується логістична функція втрат «logloss»:

$$log\,loss = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log \hat{y}_i + (1 - y_i) log\left(1 - \hat{y}_i\right)\right), \tag{4.6}$$

де N – розмір вибірки, y_i – коректна відповідь, $\hat{y}_i$ – наше передбачення.

Рівняння (4.6) може бути подано як деяке зважене «accuracy», тобто «logloss» нас сильно штрафує за впевненість у неправильних прогнозах.

4.5 Розв'язання задачі лінійної регресії

Для роботи з лінійними моделями для розв'язання двох задач регресії і класифікації потрібно використати засоби бібліотеки *sklearn* [1, 3].

Для початку потрібно імпортувати бібліотеку для візуалізації Matplotlib і *seaborn*. Також потрібно імпортувати бібліотеку pandas для роботи з бібліотекою sklearn і її модулем datasets. У datasets міститься певний набір стандартних базових dataset, на яких можна перевіряти роботу моделей, виконувати аналіз і тренування.

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

from sklearn import datasets.
```

Для початку можна розв'язати задачу регресії, тобто передбачити якусь дійсну змінну за допомогою dataset boston, в якому міститься інформація про вартість і характеристики будинків в районах міста Бостон, США. Якщо подивитися на опис даного dataset, то у ньому є різні характеристики в районах міста, і цільова змінна – це середнє значення вартості будинку в одиницях тис. доларів

boston = datasets.load_boston()
boston.keys()

print(boston.DESCR[100:1300]).

Тобто, є певна цільова змінна MEDV і різні характеристики типу CRIM_rate, тобто рівень злочинності в даному районі, вік, кількість кімнат тощо [7, 8].

Загалом, використовуючи відповідні дані потрібно вміти передбачувати цільове значення, тобто середню вартість будинку. Для початку потрібно візуально проаналізувати дані. Завжди потрібно дивитися на dataset, щоб приблизно уявляти як він виглядає (наявність пропусків, ознак у вигляді простих чисел тощо)

boston_df = pd.DataFrame(boston.data, columns=boston.feature_names)
boston_df.head().

Оскільки у нас дійсні числа, то можна розв'язувати задачу передбачення дійсної змінної за допомогою лінійної регресії. Визначаємо середню ціну будинку за допомогою *distplot* із *seaborn*

plt.figure(figsize=(6, 4))
sns.distplot(boston.target)

plt.xlabel('Price (in thousands)')
plt.ylabel('Count')
plt.tight_layout()
plt_show().

Середня ціна будинку коливається в районі 20–30 тисяч доларів. У принципі, якщо передбачання відбувається завжди за 20 або 30 тисяч доларів за цим dataset, то, швидше за все, ми вже можемо непогано вгадувати.

Оскільки навчання лінійній регресії викликає інтерес, то для цього потрібно імпортувати з модуля linear_model відповідний клас LinearRegression і отримати instance даного класу – модель, яка буде навчатися. Для того, щоб навчити модель у sklearn, потрібно викликати метод fit. Метод fit – це стандартний інтерфейс, який використовується в переважній більшості моделей. У метод fit передаються дані: Boston.data – це набір об'єктів (набір інформації про ознаки); Target – це цільова змінна (набір, список цільових змінних кожного будинку, тобто середня ціна будинку в цій області).

from sklearn.linear_model import LinearRegression

linear_regression = LinearRegression()
model = linear_regression.fit(boston.data, boston.target).

Модель пройшла навчання, і стан зберігся в змінній *model*. Для початку потрібно проаналізувати процес навчання. Відомо, що лінійна регресія «розкидає» вагові коефіцієнти з кожною ознакою для того, щоб оцінити те, який вноситься вклад в передбачення дійсної змінної. Таким чином, якщо

перемножити усі вагові коефіцієнти з відповідними ознаками, то буде отримана цільова змінна, тобто середня вартість будинку:

```
feature_weight_df = pd.DataFrame(list(zip(boston.feature_names,
model.coef_)))
feature_weight_df.columns = ['Feature', 'Weight']
print(feature_weight_df).
```

Видно, що є різні вагові коефіцієнти у різних ознак, а саме: великі і маленькі ваги. Оцінювати їх абсолютне значення не варто, тому що їх ніяк не можна нормалізувати. Проте, якщо б їх можна було б привести до якогось одного порядку, можна було б визначити їхній дійсний внесок в передбачувану здатність нашої моделі.

Оскільки лінійна регресія за функцією (4.3) перемножає вагові коефіцієнти з ознаками, то можна виконати передбачення. Причому потрібно не забути вільний коефіцієнт *intercept*, який завжди є у навченій моделі лінійної регресії:

```
import operator

first_predicted = sum(map(lambda pair: operator.mul(*pair),
    zip(model.coef_, boston.data[0])))

first_predicted += model.intercept_

print(first_predicted).
```

Отримано результат – біля 30 тисяч доларів, – що вже ближче до правди, тому що середнє значення дорівнює 20 або 30 тисяч доларів.

Для того, щоб кожен раз вручну не перемножати всі вагові коефіцієнти зі змінними ознак, використовується стандартний метод *predict*, який викликається у моделі на даних, щоб отримати конкретне передбачення. Даний метод predict автоматизує вищевказану процедуру множення вагових коефіцієнтів. Це також стандартний інтерфейс – в усіх моделях *sklearn*, крім методу *fit*, є також метод predict, який дозволяє передбачати значення.

```
predicted = model.predict(boston.data)

print(predicted[:10]).
```

Для того, щоб оцінити якість виконаної роботи (відповідність отриманого передбачення реальним значенням), можна вивести таку табличку:

```
predictions_ground_truth_df =
                pd.DataFrame(list(zip(predicted,boston.target)))
predictions_ground_truth_df.columns = ['Prediction', 'Ground truth']
predictions_ground_truth_df.head()
print(predictions_ground_truth_df).
```

Оскільки табличні значення є менш інформативними, то можна використовувати графічне подання, на якому буде відображатись передбачуване і дійсне значення.

```
plt.figure(figsize=(6, 4))
plt.scatter(predicted, boston.target)
plt.xlabel('Predicted')
plt.ylabel('Ground truth')
plt.plot([0, 50], [0, 50], color="red")
plt.tight_layout()
plt_show().
```

У дійсності на графіку всі значення мають розміщуватись на лінії під 45^0. Більш того, є певна похибка, якийсь очікуваний розкид, тому що лінійна модель (модель лінійної регресії) досить проста і слабка модель [5].

4.6 Розв'язання задачі класифікації

Розглянемо задачу класифікації на стандартному dataset, де потрібно класифікувати рак грудей на доброякісну і злоякісну пухлини [8]. Даний dataset є стандартним, який може бути завантажений за допомогою модуля datasets

```
from sklearn import datasets
cancer = datasets.load_breast_cancer()
cancer.keys().
```

для отримання опису цього dataset використовується команда:

```
print(cancer.DESCR[:760]).
```

У даному прикладі близько 30 характеристик конкретного типу пухлини, а загалом близько 600 об'єктів. На цих даних можна навчатися передбачати класифікацію (вирішувати задачу класифікації, передбачати 0 або 1, доброякісна чи злоякісна у пухлина). Проаналізуємо відповідні дані.

```
import pandas as pd
cancer_df = pd.DataFrame(cancer.data)
cancer_df.columns = cancer.feature_names
cancer_df.head()
print(cancer_df).
```

У відповідних даних є числові змінні, які визначають певні характеристики нашого об'єкта. Для того, щоб проаналізувати розподіл класів, потрібно ввести такі команди:

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(6, 4))
sns.countplot(cancer.target)

plt.xlabel('Class')
plt.ylabel('Count')
plt.tight_layout()
plt_show().
```

Переглядати поділ класів в задачах кваліфікації завжди важливо і необхідно. По-перше, потрібно виявити наявність пропусків значень. По-друге, залежність і розподіл класів між собою впливають на поведінку моделі. Завжди важливо розуміти рівномірність розподілу. У даному випадку одного класу приблизно в два рази більше, ніж інших подібних, але це не суттєво.

Наступною необхідно імпортувати дану лінійну модель – логістичну регресію із модуля *linear_model*. Для цього виконуються аналогічні дії: інстанціюється відповідний клас, після чого в інстанси викликається метод *fit*. Йому передаються відповідні дані, опис об'єктів відповідно до ознак і цільова змінна, яка у даному випадку є дискретною відповіддю «0» або «1» (доброякісна або злоякісна пухлина).

```
from sklearn.linear_model import LogisticRegression
logistic_regression = LogisticRegression()
model = logistic_regression.fit(cancer.data, cancer.target).
```

Також можна проаналізувати коефіцієнти звичайної моделі, у якій також є певні числа, тому що модель лінійна:

```
print(model.coef_).
```

Для визначення передбачення викликається метод *predict*. У даному випадку передбачається не дійсне число, а дискретна відповідь – «0» або «1»:

```
prediction = model.predict(cancer.data)
print(prediction[:10]).
```

Оскільки аналізується логістична регресія, то в даній моделі є метод *predict_proba*. Для даного типу логістичної регресії важливо аналізувати не тільки передбачений клас, а ще й наскільки модель впевнена у своєму передбаченні, наскільки вона розподіляє свою впевненість по різних класах. Наприклад, можна прогнозувати «0», але виконувати це з упевненістю «0,6», що досить сильно відрізняється від упевненості «0,9» [1, 7]. Дуже часто це потрібно для розв'язання задач багатокласової класифікації, де присутня множина відповідей (від двох і більше), яким потрібно визначити в цілому розподіл відповідей в нашій моделі,

```
prediction = model.predict_proba(cancer.data)
print(prediction[:10]).
```

Також у моделі є метод *score*, який визначає *Accuracy*:

```
print('Accuracy: {}'.format(model.score(cancer.data, cancer.target))).
```

У даному випадку процес навчання пройшов відносно непогано, що дозволило з ймовірністю 0,95 визначати наявність доброякісної чи злоякісної пухлини у даного об'єкта.

4.7 Контрольні питання

1. Основні види машинного навчання. Призначення і застосування.
2. Що таке лінійна регресія? Мета і призначення.
3. Призначення функціонала якості в задачах аналізу даних.
4. Для якого типу задач застосовують інтелект-логістичну регресію?

5 МЕТОДИ ВИЗНАЧЕННЯ ЯКОСТІ МАТЕМАТИЧНИХ МОДЕЛЕЙ

5.1 Методи вимірювання якості математичних моделей

Для підготовки даних і навчання моделі машинним навчанням підходом «з учителем» використовуються багато різноманітних методик. Однією з найпростіших методик є відправлення даних в обрану математичну модель і навчання її за допомогою якогось методу оптимізації, наприклад, за допомогою градієнтного спуску [1, 3, 8]. Після навчання дана математична модель буде характеризуватись певними параметрами, що дозволить її в подальшому використовувати. Проте потрібно перевірити ефективність роботи нашої математичної моделі на нових даних, оскільки основна мета побудови алгоритму навчання «з учителем» полягає в тому, щоб дана модель ефективно працювала не тільки на тренувальній вибірці, але й взагалі, щоб була можливість впровадження для реалізації коректних закономірних передбачень.

Оскільки перевірка коректності роботи навченої математичної моделі відбувається на нових (реальних) даних, то використання абсолютно всіх початкових даних для навчання є неефективною практикою. Тому найчастіше dataset розбивається на декілька частин, а саме: на тренувальну і тестову вибірки. На тренувальній вибірці відбувається навчання і налаштовуються параметри безпосередньо самої математичної моделі [2, 9]. Після чого відбувається перевірка роботи даної моделі на тестовій вибірці і визначається ефективність самого передбачення. Але і в даному випадку мають місце проблеми, пов'язані з наявністю ефективної роботи на тренувальній вибірці, але залишається проблема некоректної роботи на тестовій вибірці, що є результатом «перенавчання» математичної моделі на даній тренувальній вибірці. Дане явище називається «оверфіттінгом», яке описує деякі шумові залежності тренувальної вибірки, але унеможливлює опис природних залежностей, які потрібно моделювати.

Для вирішення даної проблеми існують різні підходи. Одним із таких підходів є випадкове розбиття dataset на тренувальну і тестову вибірки у такому процентному співвідношення, як 80 на 20, або 90 на 10. Однак усе залежить і від кількості об'єктів, зокрема, для великої кількості об'єктів виділяється певна тисяча, для якої і відбувається тестування. Окремою за-

дачею є передбачення за часовими ознаками або за часовими рядами. У даному випадку перемішування неможливе, тому що будуть враховуватись дані майбутніх передбачень, а це, у свою чергу, вимагає використання даних, розбитих за часом. Також якщо досліджується певний клас, який має малу кількість об'єктів, потрібно слідкувати, щоб усі об'єкти даного класу не виявилися, наприклад, у тестовій вибірці, інакше процес навчання не відбудеться.

Для вирішення проблеми «перенавчання» математичної моделі також існує багато підходів. Одним із таких підходів, у випадку лінійної регресії, є метод зменшення потужності математичної моделі (побудова полінома меншого ступеня). У даному випадку дана модель стане більш простою, а це, у свою чергу, дозволить описувати дійсну реальну залежність даних, а не якісь шумові викиди [5, 8].

Наступним підходом є метод регуляризації, який дозволяє визначити математичну модель, яка більш ефективно описує дані. Зокрема, при побудові передбачуваної моделі, відбувається пошук певної функції у великому сімействі функцій. Більше того, не існує єдиного розв'язку. Існує множина рішень (функцій), які з достатньою точністю виконують передбачення, але всі ці функції мають різні параметри, що безпосередньо впливає на складність самої математичної моделі. Тому метод регуляризації направлений на пошук максимально простої моделі (мінімальна кількість вагових коефіцієнтів, в разі лінійної регресії). Розглянемо регуляризацію *L*.2:

$$ridge\_loss = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2 + \lambda\sum_{i=1}^{k} w_i^2, \tag{5.1}$$

де N – розмір вибірки, y_i – коректна відповідь, $\hat{y}_i$ – передбачення, λ – ваговий коефіцієнт при регуляризаторі, w_i – ваговий коефіцієнт математичної моделі, k – кількість вагових коефіцієнтів.

Таким чином, додатково відбувається штрафування за велику кількість і великі значення вагових коефіцієнтів, які відповідають, відповідно, екстремальним пікам даної лінійної регресії, а це їх, у свою чергу, усуває. Таким чином відбувається пошук функції в сімействі усіх можливих функцій, що ефективно буде описувати природну залежність відповідних даних.

У свою чергу, регуляризація *L*.1 використовує модулі вагових коефіцієнтів:

$$ridge\_loss = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2 + \lambda\sum_{i=1}^{k}\left|w_i\right|, \tag{5.2}$$

де N – розмір вибірки, y_i – коректна відповідь, $\hat{y}_i$ – передбачення, λ – ваговий коефіцієнт при регуляризаторі, w_i – ваговий коефіцієнт математичної моделі, k – кількість вагових коефіцієнтів.

Регуляризація $L.1$ (5.2) дозволяє відбирати ознаки таким чином, щоб відбувалось заповнення нулями ознак, які не потрібні для даної математичної моделі. Тому регуляризація допомагає боротися з оверфіттінгом з перенавчанням, дозволяючи виділити якусь конкретну функцію в сімействі усіх визначених функцій, а також безпосередньо спрощує саму математичну модель.

Але при регулярному послідовному розбитті dataset на тренувальну і тестову вибірки з послідовним навчанням даної математичної моделі поступово відбувається перенавчання ще й під тестову вибірку. Тому що відбувається модифікація параметрів даної моделі для рівнозначної роботи на тренувальній і на тестовій вибірках. Тому найчастіше використовуються більш складні методи розбиття dataset. Першим є метод розбиття dataset на три частини: на тренувальну вибірку, на якій ми тренуємося (на якій відбувається навчання даної математичної моделі); на тестову вибірку (на якій підбираються параметри даної математичної моделі); і на валідаційну, що не береться до уваги до останнього моменту. Але при впровадженні або публікації розробленої математичної моделі сама якість її, яка в подальшому буде прийматись за основну, визначається на основі валідаційної вибірки.

Другим, не менш популярним методом розбиття є крос-валідація [3–7]. У даному методі, для невеликих dataset, уже валідаційна частина даних буде задіяна для кращого навчання математичної моделі. При використанні методу крос-валідації загальний dataset розбивається на n якихось приблизно рівних частин. Наступним відбувається послідовне використання однієї з цих частин як тестової вибірки, а інших – як тренувальної, після чого відбувається між ними обмін. Таким чином, частина даних є тестовою, а решта – тренувальними. Хоча за даним методом використовуються усі дані, але це дозволяє не проходити процес перенавчання. У результаті відбувається усереднення якості крос-валідації і використання його як метрики крос-валідації.

Одним із особливостей машинного навчання є дотримання компромісу між зміщенням і розкидом (bias-variance tradeoff). З одного боку, потрібно, щоб математична модель максимально описувала тренувальну вибірку (врахування всіх існуючих залежностей), а з іншого – потрібно уникати «перенавчання» під неї.

З одного боку, якщо побудується математична модель, яка найкраще описує тренувальну вибірку (показник variance буде високим), але при спробі використання даної математичної моделі на інших даних отримаємо великий розкид в оцінках. З іншого боку, якщо буде побудована занадто проста математична модель, яка слабо описує тренувальну вибірку, тоді як результат буде погане передбачення зі зміщенням. Таким чином відбувається певне маневрування (гра), яка називається «bias-variance tradeoff», в якій потрібно розуміти, що при навчанні математичної моделі завжди потрібно вибирати якесь «менше з двох зол».

5.2 Метрики якості математичних моделей

Оцінку якості побудованих математичних моделей машинного навчання доцільно розглянути на навченій лінійній регресії або класифікації, що дозволить визначити їхню ефективність роботи [1, 5].

Зокрема, для задачі класифікації передбачається значення «0» (модель спрацьовує) або «1» (модель не спрацьовує) і додатково може бути побудована матриця помилок, яка показує, як співвідносяться визначені передбачення з реальними даними. Наприклад, якщо математична модель спрацювала і показала значення «1», а насправді має бути «0», тоді це «false positive» (*FP*), а якщо було показано «0» там, де має бути «0» – це «true negative» (*TN*). Найпростіша і логічна метрика, яка може бути використана в задачах класифікації, це «*accuracy*» (частка правильних відповідей у математичній моделі):

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}. \tag{5.3}$$

Але, незважаючи на те, що метрика «*accuracy*» досить часто використовується, у деяких випадках абсолютно не може бути застосована, наприклад, при наявності серйозного дисбалансу класів. Припустимо, потрібно вирішити задачу класифікації рівня здоров'я пацієнтів (здоровий і нездоровий), при цьому є якась рідкісна хвороба, що зустрічається в одному випадку зі ста. І якщо буде побудовано певне константне передбачення, яке стверджує про здоровий стан людини, тоді значення «accuracy» дорівнюватиме 99%. Хоча значення величини «accuracy» достатньо високе, але дана математична модель у даному випадку непрацездатна. Для подолання подібних проблем потрібно використовувати більш складні метрики якості:

– метрика «*Precision*» показує частку коректного спрацьовування математичної моделі на рівні з усіма позитивними спрацюваннями. Наприклад, якщо показується «1», тоді наскільки часто це виконується точно

$$precision = \frac{TP}{TP + FP}; \tag{5.4}$$

– метрика «*Recall*» показує наскільки більше «1» в цілому було знайдено, і як повно були описали наші дані

$$recall = \frac{TP}{TP + FN}. \tag{5.5}$$

Дані метрики (5.4) і (5.5) дуже часто можуть використовуватись у разі дисбалансу класів. Якщо потрібно об'єднати «*precision*» і «*recall*», тоді використовують середнє гармонічне між ними:

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}. \quad (5.6)$$

Також існує параметр «logloss» (див. (4.5), підрозд. 4.4), який розглядається як певне зважене значення «accuracy», тобто параметр «logloss» суттєво штрафують за впевненість у неправильній відповіді.

І ще двома важливими конструктивними параметрами, які дуже часто використовуються, є «*ROC AUC*» і «*P-R AUC*» [3, 5, 7].

При використанні конструктивного параметра «ROC AUC» відбувається відкладання «*true positive rate*» проти «*false positive rate*»:

$$true\_positive\_rate = \frac{TP}{TP + FN}; \quad (5.7)$$

$$false\_positive\_rate = \frac{FP}{FP + TN}. \quad (5.8)$$

Необхідно відмітити, що у нас визначається площа під побудованою кривою. Таким чином, якщо дана площа дорівнює «1» (повністю закритий єдиний квадрат), то класифікатор є ідеальним. Якщо площа дорівнює приблизно «0,5», то класифікатор є випадковим і значного сенсу він немає.

Аналогічно і з класифікатором «precision-recall» кривої (1 – класифікатор ідеальний, і т. д.), яка дуже часто використовується у випадку наявності дисбалансу класів [2, 11].

Також можна отримати RMSE метрику, яка часто використовується як узагальнення метрики MSE (див. (4.2), підрозд. 4.3)

$$RMSE = \sqrt{MSE}. \quad (5.9)$$

Важливою метрикою є коефіцієнт детермінації «*R-squared*» ($-\infty \geq R^2 \geq 1$)

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \hat{y})^2}, \quad (5.10)$$

де N – розмір вибірки, y_i – значення коректної відповіді, y_i – значення передбачення, y – середнє значення вибірки.

Коефіцієнт детермінації показує, як краще можна описати дисперсію у відповідних даних. Якщо коефіцієнт детермінації дорівнює одиниці, тоді це виступає показником наявності можливості для покращення якості, але якщо показник «*R-squared*» менше нуля, тоді дана математична модель немає сенсу [9, 12].

5.3 Застосування метрик якості

З метою визначення ефективності роботи побудованої математичної моделі на нових невизначених даних потрібно оцінити якість побудованої математичної моделі і визначити метод розбиття даних [2].

Для цього доцільно використати бібліотеку sklearn і стандартний dataset, наприклад, вищевказаний dataset з раку грудей. Також додатково імпортується новий модуль metrics, в якому містяться функції, які допомагають оцінити якість роботи побудованих моделей. Для початку можна використати логістичну регресію, з якою вже знайомі

from sklearn import datasets

from sklearn import metrics

from sklearn.linear_model import LogisticRegression.

Навчання даної логістичної регресії проводиться за допомогою методу fit на всіх даних. Зважене значення метрики точності Accuracy аналізується за допомогою методу score у даній моделі

cancer = datasets.load_breast_cancer()

logistic_regression = LogisticRegression()

model = logistic_regression.fit(cancer.data, cancer.target)

print('Accuracy: {:.2f} '.format(model.score(cancer.data, cancer.target))).

Аналогічне зважене значення метрики точності Accuracy можна визначити, побудувавши модель для прогнозування за допомогою методу predict на всіх даних, і шляхом виклику у модуля metrics різних функцій. Наприклад, команда accuracy_score також повертає зважене значення Accuracy. Існує досить велика кількість різних метрик як для класифікацій, так і для регресії, які можуть бути викликаними, можуть використовуватись і аналізуватись [4, 7]

predictions = model.predict(cancer.data)

print('Accuracy: {:.2f} '.format(metrics.accuracy_score(cancer.target, predictions)))

print('ROC AUC: {:.2f} '.format(metrics.roc_auc_score(cancer.target, predictions)))

print('F1: {:.2f} '.format(metrics.f1_score(cancer.target, predictions))).

Вищевказаним було відмічено, що процес навчання на усіх даних не завжди правильний шлях, тому що неможливо достовірно оцінити поведінку даної моделі на будь-яких нових даних. Наприклад, при впровадженні побудованої моделі було б раціонально навчати моделі на одних даних, після чого аналізувати її процес поведінки на деякому новому наборі даних [8, 13]. У даному випадку можна використати метод model_selection

from sklearn.model_selection import train_test_split.

У методі train_test_split передаються відповідні дані, передаються їх ознаки і цільова змінна [3, 12]. Дані розбиваються на тестову і тренувальну

вибірку. У тренувальній вибірці, на якій відбувається навчання, підбираються параметри моделі після чого оцінка її якості виконується на тестовій вибірці і на нових об'єктах, які ще не були визначеними. Це у свою чергу дозволяє визначити ступінь навченості природної залежності у розробленій моделі. З метою фіксації якості використовується випадкове значення seed 12

```
x_train, x_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, test_size=0.2, random_state=12)
model = logistic_regression.fit(x_train, y_train)

print('Train accuracy: {:.2f}'.format(model.score(x_train, y_train)))
print('Test accuracy: {:.2f}'.format(model.score(x_test, y_test))).
```

Значення якості моделі на тестових даних дещо гірше, що є логічним результатом навчання моделі на одних даних, а визначення значення якості – на інших. Відмінність значень якості незначна, але якщо значення буде більше або дорівнювати 0.6, то це свідчить про перенавчання моделі під тренувальну вибірку [2, 3, 10].

Аналогічну процедуру можна провести і для задачі регресії. При цьому відмінностей небагато, але можна використати декілька нових класів, а також інші можливості лінійної регресії.

Із модуля linear_model 3-го класу імпортуються методи Lasso, Ridge і ElasticNet [2, 4–7], які дозволяють реалізувати єдину регресію з різними видами регуляризації. Lasso – це L_1-регуляризація, коли у нашу функцію додаються помилки першої норми ваг, тобто додаються функції суми модулів ваг, які в даний момент містяться у відповідній лінійній регресії. Ridge – це гребнева регресія, яка додає суму квадратів ваг. У ElasticNet додаються як L_1- так і L_2-регуляризація. Тому для роботи використаємо dataset boston:

```
from sklearn.linear_model import Lasso, Ridge, ElasticNet

boston = datasets.load_boston()

lasso = Lasso()
ridge = Ridge()
elastic = ElasticNet()

for model in [lasso, ridge, elastic]:
x_train, x_test, y_train, y_test = train_test_split(\
    cancer.data, cancer.target,
    test_size=0.2)
model.fit(x_train, y_train)

predictions = model.predict(x_test)
print(model.__class__)
```

print('MSE: {:.2f}\\n'.format(metrics.mean_squared_error(y_test, predictions))).

У даному випадку значення метрики MSE трохи нижче, ніж інших функцій, що свідчить про більш кращу поведінку гребневої регресії.

Також можна працювати з іншими функціямиіз модуля metrics, наприклад, L_2 score

print('R2: {:.2f} '.format(model.score(x_test, y_test)))

print('R2: {:.2f} '.format(metrics.r2_score(y_test, predictions))).

Наступний розповсюджений корисний метод – це крос-валідація, де використовується підхід, який дозволяє не відкладати деякий набір даних у чорний ящик. Це дозволяє не тільки ними користуватись для подальшого тестування, але й використовувати усі дані взагалі. Проте це використовується у тому випадку коли відомо, що дана модель не здатна до перенавчання і має високий ступінь адекватності [8, 9].

Для прикладу можна використати новий dataset-iris (стандартний), де буде вирішуватись задача класифікації на прикладі квітів типу іриси

from sklearn.model_selection import KFold, cross_val_score

iris = datasets.load_iris() iris.keys().

У наявності є чотири ознаки квітки – це довжина і ширина пелюстки, довжина і ширина чашолистника. Планується передбачення трьох різних класів ірисів, причому виявляється, що це можна реалізувати тільки за цими характеристиками. Тобто, потрібно визначити один із трьох класів: 0, 1 або 2.

Дана задача буде вирішуватись за допомогою логістичної регресії [5, 8]. Для того, щоб використовувати крос-валідацію, в циклі можна використати метод split і розбити наші дані, наприклад, на п'ять блоків – безпосередньо п'ять разів використовувати кожен з них як тестовий блок

```
logistic_regression = LogisticRegression()
cv = KFold(n_splits=5) # +StratifiedKFold

for split_idx, (train_idx, test_idx) in enumerate(cv.split(iris.data)):
    x_train, x_test = iris.data[train_idx], iris.data[test_idx]
    y_train, y_test = iris.target[train_idx], iris.target[test_idx]

    logistic_regression.fit(x_train, y_train)
    score = logistic_regression.score(x_test, y_test)
    print('Split {} Score: {:.2f} '.format(split_idx, score)).
```

У даному випадку зважене значення метрики точності Accuracy дещо варіюється, як від прийнятних, так і до непридатних.

Відповідно, логічним варіантом було б усереднювати цю якість на крос-валідації, де після чого обрати середнє значення. Проте виконувати ітерацію подібним шляхом крос-валідації не завжди раціонально, що вимагає впровадження автоматизації за допомогою функції cross_val_score:

```
cv_score = cross_val_score(
    logistic_regression, iris.data, iris.target, scoring='accuracy', cv=cv)
```

print('Cross val score: {}'.format(cv_score))
print('Mean cross val score: {:.2f}'.format(cv_score.mean())).

5.4 Контрольні питання

1. Які існують методи і підходи для вирішення проблеми «перенавчання» математичної моделі?
2. Мета і призначення метрик якості математичних моделей.
3. У яких випадках неможливо застосовувати метрику якості на основі частки правильних відповідей у математичній моделі «*accuracy*»?
4. Що таке коефіцієнт детермінації при визначення якості математичних моделей?

6 АНСАМБЛЕВІ МАТЕМАТИЧНІ МОДЕЛІ

6.1 Модель на основі дерева прийняття рішень

Основним недоліком методів лінійних класифікації та регресії є сфера застосування їх для лінійних моделей на основі використання лінійних залежностей [1, 4, 13]. Більше того, ці підходи значно відрізняються від реальних принципів загального прийняття рішень людиною. Прикладом такого прийняття рішення може слугувати діагнозу лікарем на основі набору послідовно пов'язаних запитань хворій людині. Такий підхід визначення діагнозу хворої людини базується на математичній моделі в основі якої є використання дерево прийняття рішень. У математичній моделі дерева прийняття рішень на основі певних критеріїв і набору контекстних запитань поступово відбувається визначення значення елементарної складової. У разі класифікації елементарна складова буде визначати клас, а регресії – діапазон [2, 7].

Модель дерева прийняття рішення подається як алгоритм, що послідовно розбиває простір за різними критеріями з метою максимізації кількості схожих об'єктів у піддереві (в лівому або правому), нормуючи при цьому дані об'єкти за кількістю. Процес побудови нашого дерева відбувається до кінця (зазначеної глибини), до ступеня дискретизації відповіді, при цьому елементарна складова дерева являє собою клас, що зустрічається найчастіше, або якесь значення (у випадку регресійного аналізу).

Модель на основі дерева прийняття рішення зазвичай використовується в композиції (ансамблі), тому що дані моделі мають властивість до перенавчання [3, 6]. Використання будь-якої окремо побудованої глибокої моделі дерева прийняття рішення не є раціональним через перенавчання на тренувальній вибірці, що, у свою чергу, зробить неможливим її використання в подальшому.

6.2 Модель прийняття рішень на основі випадкового лісу

Ансамблевий метод машинного навчання випадковий ліс (random forest) базується на композиції чисельних моделей дерев прийняття рішень.

Принцип роботи методу випадковий ліс базується на загальному принципі роботи усіх ансамблевих моделей. У випадку ансамблевої моделі будується велика кількість слабких різних базових алгоритмів з подальшим їх сумісним використанням (усереднення результату).

Самі по собі ці базові алгоритми не є ефективними, оскільки вони дуже перенавчені і мають значний розкид результатів прогнозування, що не дозволяє їх окремо використовувати, але у своїй сукупності вони дають досить точні передбачення. Наприклад, є відома історія [8, 13] про ярмарок у XIX столітті, коли вісімсот селян намагалися здійснити припущення про вагу свійської тварини – бика – на ярмарку. Проте ніхто з присутніх селян так і не змігдати правильної відповіді, але якщо усереднити всі їх передбачення, то виявиться, що отримане число з точністю до одного фунта наближається до дійсної ваги цієї тварини. Вищенаведений приклад є принципом роботи композиційних методів. Даний метод базується на усередненні великої кількості різних передбачень, які, самі по собі, мають сильний розкид, але у подальшому використовуються як досить сильний підсумковий алгоритм.

При побудові випадкового лісу паралельно будується велика кількість різних дерев прийняття рішень. Для підтримки різноманітності дерев прийняття рішень може бути обрана якась підмножина об'єктів або ознак. При цьому об'єкти, наприклад, можуть бути обрані за допомогою алгоритму, який називається bootstrap; а саме: якщо тренувальна вибірка має розмір m, тоді з неї може бути отримано n об'єктів з поверненням. Таким чином, кожен раз, коли отримується і залишається об'єкт у тій же вибірці, тоді нічого не заважає з ним працювати ще раз. У дану використовувану тренувальну вибірку потрапляє приблизно 63% об'єктів, на яких відбувається навчання. Решта об'єктів може використовуватись для валідації, що, у свою чергу, називається out-of-bag [7, 9].

Таким чином паралельно будуються глибокі дерева прийняття рішень з метою отримання великого розкиду значень при класифікації. Оскільки процес побудови дерев прийняття рішень є незалежним, то такі різні процеси, як навчання і прийняття рішень, можна виконати за допомогою паралельного обчислення на різних автономних комп'ютерних системах.

При побудові дерева прийняття рішення, а саме: при розбитті на різну кількість ознак у листочках і вершинах дерева можуть використовуватись різні випадкові підмножини ознак.

Також потрібно зауважити, що модель прийняття рішень на основі випадкового лісу дуже рідко здатна до перенавчання. Таким чином відбувається побудова багатьох різних дерев прийняття рішень з подальшим їх усередненням, що загалом підвищує якість результатів класифікації і про-

гнозування. Але є й недоліки, які полягають у трудомісткості процесу побудови глибоких дерев прийняття рішень з метою отримання великого розкиду результатів класифікації і прогнозування [2, 10].

6.3 Метод градієнтного підсилення

Відомо, що у моделі прийняття рішення на основі випадкового лісу процес побудови базових алгоритмів і дерев прийняття рішень є паралельним (незалежним), а це, у свою чергу, дозволяє сам процес навчання проводити за допомогою різних автономних комп'ютерних систем [2, 8].

У випадку застосування метаалгоритму машинного навчання типу бустінг базові алгоритми стають залежними, а це, у свою чергу, кожному наступному алгоритму дозволяє виправляти помилки існуючої моделі. Таким чином до якогось базового алгоритму додається новий, що спричиняє виправлення помилки існуючого і т. д. Дана процедура виконується до тих пір, поки підсумкова якість моделі класифікації і прогнозування не задовольнить користувача [1, 8].

На відміну від моделі прийняття рішень типу випадковий ліс для методу градієнтного бустінга, при роботі над деревами прийняття рішень не обов'язково будувати структуру регресійного дерева глибоким і до кінця.

Достатньо декілька невеликих коротких дерев прийняття рішень. Вони суттєво впливають на загальну продуктивність і продуктивність алгоритму при навчанні моделі у цілому, а це, на рівні з моделлю прийняття рішення типу випадковий ліс, вимагає побудови значної кількості дерев прийняття рішень із глибокою структурою. Таким чином, незважаючи на те, що модель прийняття рішень на основі методу випадковий ліс має задовільну якість і її процес навчання досить складний [3, 7].

Методу машинного навчання на основі градієнтного бустінга для роботи достатньо декілька неглибоких дерев, які поступово додаються до нашого базового алгоритму. Таким чином можуть додаватись наприклад, нові базові алгоритми з таким кроком, щоб відповідна модель не перенавчалась. Проте метод градієнтного бустінга, на відміну від моделі прийняття рішень типу випадковий ліс, має властивість до перенавчання, що вимагає процесу стеження за тим, щоб він не перенавчався за допомогою задіяння різного роду регуляризації (крок або стохастичний градієнтний бустінг).

6.4 Методика застосування ансамблевих моделей

Для роботи з ансамблевими моделями, а саме: з моделями прийняття рішень типу випадковий ліс і градієнтний бустінг, можна використовувати інструменти бібліотеки sklearn. Для візуалізації імпортується відповідна бібліотека, і для роботи з даними завантажується dataset iris з модуля datasets. У даному прикладі буде вирішуватись задача класифікації, тобто, визначатись вид квітки за характеристиками її пелюсток:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns from sklearn
import datasets

iris = datasets.load_iris().
```

З метою аналізу розподілу відповідних характеристик в різних класах потрібно побудувати pairplot за допомогою бібліотеки seaborn

```
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_df['Species'] = np.array([iris.target_names[cls] for cls in iris.target])
sns.pairplot(iris_df, hue='Species').
```

Результати аналізу показують, що отриманий клас setosa (синій колір) лінійно роздільний. Також може використовуватись навіть деяка лінійна модель з метою визначення належності даної квітки саме до цього класу. Проте зелений і червоний класи набагато більш схожі, а це вимагає використання деякої нелінійної залежності і побудови більш складної моделі, наприклад, деревах прийняття рішень.

Для розв'язання задачі кваліфікації потрібно імпортувати з модуля sklearn.metrics ансамблеву модель RandomForestClassifier. Проте можна й за допомогою моделі типу випадковий ліс розв'язувати задачу регресії, для використання якої є в наявності відповідний клас у даному модулі [3, 9]. Також з модуля metrics потрібно імпортувати accuracy_score і confusion_matrix, які також будуть використовуватись. Для задачі розбиття використовується стандартний метод train_test_split

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split.
```

Наступним потрібно, щоб відповідний клас був дистанційованим. Для цього потрібно застосувати модель прийняття рішення типу випадковий ліс, що містить біля ста субмоделей дерев прийняття рішень, в яких фіксується деяке випадкове значення seed

```
random_forest = RandomForestClassifier(n_estimators=100,
        random_state=42)

 x_train, x_test, y_train, y_test = train_test_split(
     iris.data, iris.target,
     test_size=0.3, stratify=iris.target, random_state=42)
 rf_model = random_forest.fit(x_train, y_train).
```

Потрібно відмітити, що з'являється новий параметр stratify. Дуже важливо, щоб при розв'язанні задачі класифікації, а саме: при розбитті зберігався розподіл класів таким чином, щоб не відбувалось навчання на даних, в яких, наприклад, відсутній якийсь один клас, який потім з'являється в те-

стовій вибірці. У такому випадку неможливо визначити залежність у цьому класі, тому необхідно, щоб співвідношення класів зберігалося при розбитті. Аналогічна умова має виконуватись під час виконання кросвалідації [5, 9].

Прогнозування або класифікація виконується на тестовій вибірці і аналізується зважене значення Accuracy за допомогою стандартної метрики accuracy_score

predictions = rf_model.predict(x_test)

print('Accuracy: {:.2f} '.format(accuracy_score(y_test, predictions))).

Потрібно проаналізувати якість отриманих результатів прогнозування або класифікації [5]. Для цього визначається метрика confusion_matrix зі стандартного модуля metrics і відображається за допомогою seaborn hit map:

confusion_scores = confusion_matrix(y_test, predictions)

confusion_df = pd.DataFrame(confusion_scores,
columns=iris.target_names, index=iris.target_names)

sns.heatmap(confusion_df, annot=True).

Отриманий результат типу setosa є «синім» класом, який досить добре піддається класифікації і в якому неможливо помилитись. Проте інші два класи дещо змішуються через взаємну схожість.

Також за допомогою моделей, які базуються на деревах прийняття рішень, завжди дуже легко за допомогою методу feature_importance визначати найбільший внесок у визначення тієї чи іншої ознаки (найбільша передбачувана здатність)

feature_importance = list(zip(iris.feature_names,
rf_model.feature_importances_))

feature_importance_df = pd.DataFrame(feature_importance,
columns=['Feature', 'RFImportance'])

feature_importance_df.

У даному випадку результати класифікації за допомогою методу прийняття рішення типу випадковий ліс показують, що найкраще визначення класу квітки виконується на основі критерію ширини пелюстки.

Звичайно, у моделей існує набагато більше параметрів, ніж було використано вище. Можна задіювати і незадіювати використання алгоритму bootstrap, обмежувати глибину структури регресійного дерева, змінювати кількість моделей на основі дерев прийняття рішення тощо. Наприклад, доцільно завжди визначати якість (oob_score) тих об'єктів, які не використовувалися при навчанні даних моделей на основі дерев прийняття рішень:

rf_model.get_params().

Схожим чином може бути використаний метод градієнтного бустінгу. Даний метод також вирішує задачі класифікації, і може бути використаний у задачах регресії. Оскільки використовується датасет для квіток типу ірис

і вирішується задача класифікації, то процес навчання моделей прийняття рішень виконується для дерев у кількості ста штук методом train, після чого перевіряється якість результатів розв'язання на тесті:

```
from sklearn.ensemble import GradientBoostingClassifier
gradient_boosting = GradientBoostingClassifier(n_estimators=100,
                                               random_state=42)
gb_model = gradient_boosting.fit(x_train, y_train)

print('Accuracy: {:.2f}'.format(gb_model.score(x_test, y_test))).
```

У даному випадку отримана якість результатів розв'язання трішки краща. Також додатково може бути проаналізований показник якості за допомогою методу градієнтного бустінга, що визначає вплив різних ознак

```
feature_importance_df['GB Importance'] =
gb_model.feature_importances_feature_importance_df.
```

У даному випадку видно, що якість розподіляється трішки по-іншому, а ознаки мають трішки іншу важливість, що є логічним через інший тип математичної моделі.

До важливих параметрів можна віднести декілька принципово важливих моментів, зокрема, не можна визначити, що глибина структури регресійного дерева стандартна, яких є усього три види. Тому в основі методу градієнтного бустінга і передбачається побудова неглибоких структур регресійних дерев прийняття рішень. Додатково можна варіювати й іншими параметрами

```
gb_model.get_params().
```

6.5 Контрольні питання

1. Яке призначення і особливості використання математичної моделі на основі дерева прийняття рішення в задачах задач регресії і класифікації?
2. Який принцип роботи моделі прийняття рішень на основі випадкового лісу?
3. Які є засоби реалізації методу градієнтного підсилення в задач регресії і класифікації?

ТЕМАТИКА ЛАБОРАТОРНИХ РОБІТ

Лабораторна робота № 1. Засоби реалізації математичного моделювання та аналізу даних

Мета роботи

Надбання практичних навичок із застосування основних бібліотек NumPy, SciPy і Pandas для ефективного розв'язання широкого кола задач аналітичного аналізу даних в програмному середовищі Python.

Завдання на підготовку

Студент має знати:

– поняття та особливості інтелектуальної задачі аналізу даних;

– форми та методи подання інтелектуальних задач аналізу даних;

– мови програмування PYTHON;

– основні поняття при роботі з багатовимірними масивами даних бібліотеки NumPy;

– основні поняття з подготовки та аналізу даних у бібліотеці бібліотеці Pandas;

– методи групування і перетворення ознак даних;

– основні засоби при роботі з декількома таблицями.

Студент має вміти:

– створювати програми мовою PYTHON.

Для допуску до виконання роботи потрібно:

– вміти відповісти на теоретичні питання за ходом виконання роботи;

– показати викладачу заготовку звіту про лабораторну роботу, яка має містити титульний лист та опис обраної предметної галузі.

Завдання на лабораторну роботу

1. На сайті https://www.kaggle.com/datasets підібрати будь-який датасет, що містить числові значення на рівні з текстовими даними. Із отриманим датасетом виконати усі операції на прикладі заданого датасету *citibike.csv*, що наведені в пункті 1.3.2 для роботи з об'єктом Pandas.DataFrame.

2. Для отриманого датасету за допомогою інструментів бібліотеки Pandas виконати операції: групування даних (див. підрозд. 1.4) і перетворення ознак (див. підрозд. 1.6).

3. На сайті https://www.kaggle.com/datasets підібрати три будь-яких датасетів, що містять деякі чисельні значення на рівні з текстовими даними. Із отриманими датасетами виконати усі операції на прикладі заданого датасету *boston-airbnb-open-data,* що наведений в підрозділі 1.5. для роботи з декількома таблицями (див. підрозд. 1.5).

Порядок виконання роботи

1. Встановити дистрибутив програмного забезпечення за веб-адресою https://www.anaconda.com/distribution/.

2. У встановленому Anaconda.Navigator запустити Jupyter.Notebook.

3. У робочому вікні Jupyter.Notebook, на основі наведених прикладів роботи в бібліотеці NumPy, створити два двовимірних масиви розмірами $(n+2)\times(n+2)$, які заповнені випадково згенерованими числами в діапазоні від 0 до 9 (n – порядковий номер студента у списку журнала).

Над отриманими матрицями виконати операції додавання, віднімання, ділення, а також скалярне і векторне множення. Для отриманих матриць, за результатами арифметичних операцій, знайти: визначник, власні числа, власні вектори.

4. У бібліотеці SciPy для заданої функції (табл. 1.1) визначити: похідну (в заданій точці), первісну (на заданому проміжку), а також виконати оптимізацію (визначити глобальний і локальний мінімуми).

5. У бібліотеці Pandas створити одновимірний масив із вільно заданими мітками безпосередньо самим студентом розміром $n+1$ (n – порядковий номер студента у списку журналу). Також у даній бібліотеці створити словник розміром $n+1$ із відповідними індексами, вільно заданими студентом. Над даним одновимірним масивом виконати операції, наведені у підпункті 1.1.3.1 для роботи з об'єктами Pandas.Series.

Зміст звіту

1. Звіт з лабораторної роботи має бути виконаний на листах формату А4.

2. Звіт має містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.

3. В розділі «Результати виконання лабораторної роботи» студент має додати роздруковані на принтері лістінг програми з коментарями і результати роботи в бібліотеках NumPy, SciPy і Pandas, проілюстрований скріншотами, на основі завдань, наведених у даній лабораторній роботі, а також написати висновки з виконання даної лабораторної роботи.

Приклади завдань на лабораторну роботу

Таблиця 1.1 – Варіанти завдань

Ч.ч.	Функція $f(x)$	Інтервал інтегрування $[a; b]$	Координати точки для похідної функції $f(x)$
1	$\ln x / x\sqrt{1+\ln x}$	[1,0; 3,5]	1,0
2	$tg^2 x + ctg^2 x$	[π/6; π/3]	π/4
3	$1/x\ln x$	[1,5; 3,0]	2,0
4	$\ln^2 x / x$	[1,0; 4,0]	2,0
5	$\sqrt{e^x - 1}$	[0; ln2]	1,0

Продовження табл. 1

6	$xe^x \sin x$	[1,0; 4,0]	2,0
7	$0,5x(e^x - e^{-x})$	[0; 2,0]	1,0
8	$1/\sqrt{9+x^3}$	[2,0; 5,0]	3,0
9	$\sin(1/x)x^4$	[1,0; 2,5]	2,0
10	$x^3 arctgx$	[0; $\sqrt{3}$]	1,0
11	$\arcsin\sqrt{\frac{x}{(1+x)}}$	[0; 3,0]	2,0
12	$x^2(1+\ln x)$	[1,5; 3,0]	2,0
13	$1/\sqrt{1+3x+2x^2}$	[0; 5,0]	2,0
14	$\sqrt{x^2-0,14}/x$	[2,3; 6,0]	2,0
15	$2^{3x}\ln\lvert\cos x\rvert$	[0; π/2]	π/6
16	$(e^{3x}+1)/(e^x+1)$	[0; 2,0]	3,0
17	$xarctgx/\sqrt{1+x^2}$	[0; 2,0]	2,0
18	$\sin^5 x/\ln(1+x^3)$	[0; π/4]	2,0
19	$x^2\sqrt{4-x^2}$	[0; 1,8]	1,0
20	$e^x \cos^2 x$	[0; 2π]	π/8
21	$\sin x / arcctg8x$	[0; π]	π/3
22	$\text{arcctg}3x + 2\text{arcctg}x$	[0; 1,5π]	π/5
23	$7x + 6\text{arctg}x$	[0; 3π]	π/7
24	$x - 4\text{arcctg}x + \sin x$	[0; π]	π/8
25	$xtg5x - 4\text{arcctg}x + \sin x$	[0; π/2]	π/10
26	$x^2 + \ln x + 2$	[2,5; 5,0]	2,0
27	$e^x + \cos^2 x + x^3$	[-2π; 2π]	π/8
28	$\text{arcctg}2x + 5\text{arcctg}x - \ln x$	[-2π; 2π]	π/5
29	$\ln x/\sqrt{1+x+x^3}$	[-5,0; 5,0]	2,0
30	$x + e^x \sin 2x + \ln x$	[1,0; 4,0]	2,0

Лабораторна робота № 2. Методи та засоби візуалізації результатів аналізу даних

Мета роботи

Надбання практичних навичок з візуалізації результатів аналітичного аналізу даних за допомогою бібліотек Matplotlib і Pandas в програмному середовищі Python для ефективного розв'язання широкого кола задач.

Завдання на підготовку

Студент має знати:

– поняття та особливості інтелектуальної задачі аналізу даних;

– форми та методи подання інтелектуальних задач аналізу даних;

– мови програмування PYTHON;

– основні поняття при візуалізації даних в бібліотеках Matplotlib, Pandas і Plotly;

Студент має вміти:

– створювати програми мовою PYTHON.

Для допуску до виконання роботи потрібно:

– вміти відповісти на теоретичні питання за ходом виконання роботи;

– показати викладачу заготовку звіту про лабораторну роботу, яка має містити титульний лист та опис обраної предметної області.

Завдання на лабораторну роботу

1. Написати програму статичної візуалізації даних за допомогою графіків із використанням засобів бібліотек Matplotlib, Pandas.

2. Написати програму інтерактивної візуалізації за допомогою вбудованих елементів бібліотеки візуалізації даних Plotly, браузера і JS-бібліотеки.

Порядок виконання роботи

1. У робочому вікні Jupyter.Notebook, на основі наведених прикладів роботи в пунктах 2.1.1 і 2.1.2 освоїти основні команди візуалізації даних в бібліотеці Matplotlib.

2. На сайті https://www.kaggle.com/datasets підібрати будь-який датасет, що містить числові значення на рівні з текстовими даними. Із отриманим датасетом виконати усі операції на прикладі заданого датасету *titanic.csv*, що наведені в пункті 2.1.3 для роботи з візуалізації даних в Pandas.

3. На сайті https://www.kaggle.com/datasets або на інших сайтах підібрати будь-який датасет, що містить числові значення на рівні з текстовими даними (економічні, технічні та соціальні характеристики будь-яких об'єктів статистичних досліджень) Із отриманим датасетом виконати усі операції на прикладі заданого датасету *life-expectancy-per-GDP-2007.csv*', що наведений в підрозділі 2.4 для роботи з інтерактивною візуалізацією з Plotly.

Зміст звіту

1. Звіт з лабораторної роботи має бути виконаний на листах формату А4.

2. Звіт має містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.

3. В розділі «Результати виконання лабораторної роботи» студент має додати роздруковані на принтері лістінг програми з коментарями і результати роботи в бібліотеках Matplotlib, Pandas, Plotly, проілюстрований скріншотами, на основі завдань, наведених у даній лабораторній роботі, а також написати висновки з виконання даної лабораторної роботи.

Лабораторна робота № 3. Застосування статистичних методів та засобів в задачах аналізу даних

Мета роботи

Надбання практичних навичок із застосування основних методів математичної статистики, реалізованих в бібліотеках NumPy, SciPy для ефективного розв'язання широкого кола задач аналітичного аналізу даних в програмному середовищі Python.

Завдання на підготовку

Студент має знати:

- поняття та особливості інтелектуальної задачі аналізу даних;
- форми та методи подання інтелектуальних задач аналізу даних;
- мови програмування PYTHON;
- основні положення теорії ймовірності;
- основні поняття при реалізації статистичних методів в бібліотеці SciPy.

Студент має вміти:

- створювати програми мовою PYTHON.

Для допуску до виконання роботи потрібно:

- вміти відповісти на теоретичні питання за ходом виконання роботи;
- показати викладачу заготовку звіту про лабораторну роботу, яка має містити титульний лист та опис обраної предметної області.

Завдання на лабораторну роботу

1.Написати програму з отримання довірчого інтервалу на прикладі заданого датасету і виконайте перевірку висунутих гіпотез.

Порядок виконання роботи

1. У робочому вікні Jupyter.Notebook, на основі наведених прикладів роботи в підрозділі 3.1, засвоїти використання методів математичної статистики за допомогою бібліотеки SciPy.

2. На сайті https://www.kaggle.com/datasets підібрати будь-який датасет, що містить статистичні дані. Із отриманим датасетом виконати усі операції для отримання довірчого інтервалу на прикладі заданого датасету *mex_clean.csv*, що наведений в підрозділі 3.2 для роботи з об'єктом Pandas.DataFrame.

3. Для отриманого датасету, що містить статистичні дані, за допомогою наведеного в підрозділі 3.3 прикладу виконати перевірку гіпотез і розподіл Стьюдента.

Зміст звіту

1. Звіт з лабораторної роботи має бути виконаний на листах формату А4.

2. Звіт має містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.

3. В розділ «Результати виконання лабораторної роботи» студент має додати роздруковані на принтері лістінг програми з коментарями і результати роботи в бібліотеці SciPy з роботи з методами математичної статистики, проілюстрований скріншотами, а також результати виконання завдань, наведених у даній лабораторній роботі.

Лабораторна робота № 4. Використання лінійних моделей в задачах інтелектуального аналізу даних

Мета роботи

Надбання практичних навичок розв'язання задач регресії і класифікації, в лінійних моделях аналітичного аналізу даних за допомогою методів і засобів машинного навчання, реалізованих в бібліотеках Sklearn для ефективного розв'язання широкого кола задач в програмному середовищі Python.

Завдання на підготовку

Студент має знати:

- поняття та особливості інтелектуальної задачі аналізу даних;
- форми та методи подання інтелектуальних задач аналізу даних;
- мови програмування PYTHON;
- основні види методів машинного навчання і регресій;
- основні поняття фунціоналу якості і градієнтного спуску при розв'язанні задач регресії і класифікації.

Студент має вміти:

- створювати програми мовою PYTHON.

Для допуску до виконання роботи потрібно:

- вміти відповісти на теоретичні питання за ходом виконання роботи;
- показати викладачу заготовку звіту про лабораторну роботу, яка має містити титульний лист та опис обраної предметної області.

Завдання на лабораторну роботу

1. Написати програму для регресійного аналізу на основі даних засобами бібліотеки Sklearn.

2. Написати програму для класифікації об'єктів за характеристичними ознаками на основі даних засобами бібліотеки Sklearn і Pandas.

Порядок виконання роботи

1. На сайті https://www.kaggle.com/datasets або на інших сайтах підібрати будь-який датасет, що містить числові характеристики певних ознак досліджуваного об'єкта (економічні, технічні або соціальні характеристики будь-яких об'єктів статистичних досліджень). Визначіть цільову змінну, яка залежить від певних ознак, описаних в обраному датасеті. Із отриманим датасетом виконати усі операції розв'язання задачі лінійної регресії (передбачення значення цільової змінної від заданих ознак), наведеної у пункті 4.1.5, на прикладі стандартного датасету boston бібліотеки Python.

2. На сайті https://www.kaggle.com/datasets або на інших сайтах підібрати будь-який датасет, що містить числові характеристики певних ознак досліджуваного об'єкта (економічні, технічні або соціальні характеристики будь-яких об'єктів статистичних досліджень). Для отриманого датасету потрібно визначити два основних типи класу, за якими буде відбуватись класифікація ознак об'єкта. З отриманим датасетом виконати усі операції розв'язання задачі класифікації, наведеної у підрозділі 4.6, на прикладі стандартного датасету *breast_cancer* бібліотеки Python.

Зміст звіту

1. Звіт з лабораторної роботи має бути виконаний на листах формату А4.

2. Звіт має містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.

3. В розділ «Результати виконання лабораторної роботи» студент має додати роздруковані на принтері лістінг програми з коментарями і результати роботи в бібліотеці Sklearn, проілюстровані скріншотами, на основі завдань, наведених у даній лабораторній роботі.

Лабораторна робота № 5. Застосування методів визначення якості математичних моделей в задачах інтелектуального аналізу даних

Мета роботи

Надбання практичних навичок із розв'язання задач регресії і класифікації з використанням метрик якості математичних моделей в лінійних моделях аналітичного аналізу даних за допомогою методів і засобів машинного навчання реалізованих в бібліотеках Sklearn і Metrics для ефективного розв'язання широкого кола задач в програмному середовищі Python.

Завдання на підготовку

Студент має знати:

- поняття та особливості інтелектуальної задачі аналізу даних;
- форми та методи подання інтелектуальних задач аналізу даних;
- мови програмування PYTHON;
- основні види методів машинного навчання і регресій;
- основні метрики і методи вимірювання якості математичних моделей;

Студент має вміти:

– створювати програми мовою PYTHON.

Для допуску до виконання роботи потрібно:

– вміти відповісти на теоретичні питання за ходом виконання роботи;

– показати викладачу заготовку звіту про лабораторну роботу, яка має містити титульний лист та опис обраної предметної області.

Завдання на лабораторну роботу

1.Написати програму для прогнозування на основі даних за допомогою різних методів машинного навчання.

2. Написати програму для визначення ефективності роботи побудованої математичної моделі на основі метрик оцінювання.

Порядок виконання роботи

1. На сайті https://www.kaggle.com/datasets або на інших сайтах підібрати будь-який датасет, що містить чисельні характеристики певних ознак досліджуваного об'єкта (економічні, технічні або соціальні характеристики будь-яких об'єктів статистичних досліджень). На основі підходів різних видів задач регресії побудувати математичну модель прогнозування за допомогою різних методів навчання і визначити ефективність роботи побудованої математичної моделі (ступінь навченості природної залежності) на основі метрик оцінювання.

Зміст звіту

1. Звіт з лабораторної роботи має бути виконаний на листах формату А4.

2. Звіт має містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.

3. В розділі «Результати виконання лабораторної роботи» студент має подати роздруковані на принтері лістінг програми з коментарями і результати роботи в бібліотеці Sklearn, проілюстровані скріншотами, на основі завдань, наведених у даній лабораторній роботі.

Лабораторна робота № 6. Використання ансамблевих математичних моделей у задачах інтелектуального аналізу даних

Мета роботи

Надбання практичних навичок із розв'язання задач регресії і класифікації на основі ансамблевих моделей за допомогою методів і засобів машинного навчання, реалізованих в бібліотеках Sklearn для ефективного розв'язання широкого кола задач в програмному середовищі Python.

Завдання на підготовку

Студент має знати:

– поняття та особливості інтелектуальної задачі аналізу даних;
– форми та методи подання інтелектуальних задач аналізу даних;
– мови програмування PYTHON;
– основні види методів машинного навчання і регресій;
– основні види ансамблевих моделей для прийняття рішень;
– основний принцип методу градієнтного підсилення.

Студент має вміти:

– створювати програми мовою PYTHON.

Для допуску до виконання роботи потрібно:

– вміти відповісти на теоретичні питання за ходом виконання роботи;
– показати викладачу заготовку звіту про лабораторну роботу, яка має містити титульний лист та опис обраної предметної області.

Завдання на лабораторну роботу

1.Написати програму для роботи з ансамблевими моделями на основі даних, за допомогою яких буде розв'язуватись задача класифікації (визначатись тип об'єкта за його характеристиками).

Порядок виконання роботи

1. На сайті https://www.kaggle.com/datasets або на інших сайтах підібрати будь-який датасет, що містить чисельні характеристики певних ознак досліджуваного об'єкта (економічні, технічні або соціальні характеристики будь-яких об'єктів статистичних досліджень). На основі різних підходів реалізації ансамблевих моделей методів машинного навчання розв'язати задачу класифікації і визначити ефективність роботи побудованої математичної моделі на основі метрик оцінювання.

Зміст звіту

1. Звіт з лабораторної роботи має бути виконаний на листах формату А4.

2. Звіт має містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.

3. В розділі «Результати виконання лабораторної роботи» студент має додати роздруковані на принтері лістінг програми з коментарями і результати роботи в бібліотеці Sklearn, проілюстровані скріншотами, на основі завдань, наведених у даній лабораторній роботі.

ЛІТЕРАТУРА

1. Уэс М. Python и анализ данных / пер. с англ. Слинкин А. А. М. : ДМК Пресс, 2015. 482 с.

2. Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение. СПб. : Питер, 2018. 576 с.: ил.

3. Копец Д. Классические задачи Computer Science на языке Python. СПб. : Питер, 2020. 256 с.

4. Брюс П., Брюс Э. Практическая статистика для специалистов Data Science / пер. с англ. СПб. : БХВ-Петербург, 2018. 304 с.

5. Силен Д., Мейсман А., Али М. Основы Data Science и Big Data. Python и наука о данных. СПб. : Питер, 2017. 336 с.

6. Грае Дж. Data Science. Наука о данных с нуля / пер. с англ. СПб. : БХВ-Петербург, 2017. 336 с.: ил.

7. Y. Ivanchuk, K. Koval, A. Halianovska. The algorithm for simulation of a nonlinear dynamic Lorence System. Proc. XII International Scientific-Practical Conference «Internet-Education-Science-2020» dedicated to the 25-th anniversary of the Computer Science Department, 2020 May 26–29, P. 123–124.

8. Моделювання та оптимізація систем : підручник / Дубовой В. М., Квєтний Р. Н., Михальов О. І., Усов А. В. Вінниця : ПП «ТД «Едельвейс», 2017. 804 с.

9. Месюра В. І., Ваховська Л. М. Основи проектування систем штучного інтелекту. Лабораторний практикум. Частина 1. Способи подання задач і пошуку розв'язків. Лабораторний практикум. Вінниця : ВНТУ, 2009. 108 с.

10. Polhul T., Yarovyi A. «Development of a method for fraud detection in heterogeneous data during installation of mobile applications». (2019) Eastern-European Journal of Enterprise Technologies, 1 (2–97), pp. 65–75. – DOI: https://doi.org/10.15587/1729-4061.2019.155060.

11. Польгуль Т. Д., Яровий А. А. Аналіз різнорідних даних в інтелектуальних системах виявлення шахрайства. Вісник Вінницького політехнічного інституту. 2019. № 2 (143). С. 78–90. – DOI: https://doi.org/10.31649/1997-9266-2019-143-2-78-90.

12. M. Granik, V. Mesyura and A. Yarovyi, «Determining Fake Statements Made by Public Figures by Means of Artificial Intelligence,» 2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT), Lviv, 2018, pp. 424–427. – DOI: https://doi.org/ 10.1109/STC-CSIT.2018.8526631.

13. Яровий А. А. Методи та засоби організації високопродуктивних паралельно-ієрархічних обчислювальних систем із рекурсивною архітектурою : монографія. Вінниця : ВНТУ, 2016. 363 с.

Навчальне видання

Іванчук Ярослав Володимирович
Месюра Володимир Іванович
Яровий Андрій Анатолійович
Манжілевський Олександр Дмитрович

ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ ТА МАШИННЕ НАВЧАННЯ
ЧАСТИНА 1
БАЗОВІ МЕТОДИ ТА ЗАСОБИ АНАЛІЗУ ДАНИХ

Навчальний посібник

Рукопис оформив *Я. Іванчук*

Редактор *В. Дружиніна*

Оригінал-макет підготовила *Т. Криклива*

Підписано до друку 03.11.2021 р.
Формат 29,7×42¼. Папір офсетний.
Гарнітура Times New Roman.
Друк різографічний. Ум. друк. арк. 4,14.
Наклад 50 (1-й запуск 1–21) пр. Зам. № 2021-114.

Видавець та виготовлювач
Вінницький національний технічний університет,
інформаційний редакційно-видавничий центр.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95,
м. Вінниця, 21021.
Тел. (0432) 65-18-06.
press.vntu.edu.ua;
E-mail: kivc.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01. 07.2009 р