

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КИЇВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Кафедра програмних систем і технологій



ІРИНА ЮРЧУК

**ПРОГРАМНІ ТЕХНОЛОГІЇ МАШИННОГО
НАВЧАННЯ**

Лабораторний практикум для студентів спеціальності
121 «Інженерія програмного забезпечення»
освітнього рівня «Магістр»

Київ 2023

Рецензенти:

к.ф-м.н., доцент, завідувач кафедри загальної математики та методики навчання інформатики факультету інформаційних технологій Волинського національного університету імені Лесі Українки МАРІЯ ХОМЯК

к.т.н., доцент кафедри мережових та інтернет технологій факультету інформаційних технологій ОКСАНА ГЕРАСИМЕНКО

Рекомендовано до публікації кафедрою програмних систем і технологій протокол №4 від «19» жовтня 2023 року.

Схвалено науково-методичною комісією факультету інформаційних технологій (протокол № 5 від 25.03.2024 року).

ІРИНА ЮРЧУК

Програмні технології машинного навчання [Електронний ресурс]: лабораторний практикум / Ірина Юрчук. – К. : КНУ імені Тараса Шевченка, 2023. – 43с.

Містить завдання, методичні вказівки та варіанти для виконання лабораторних робіт. Складено відповідно до робочої програми освітньої компоненти «Програмні технології машинного навчання».

Для студентів спеціальності 121 «Інженерія програмного забезпечення» освітнього рівня «Магістр».

Публікується в авторській редакції.

Електронна версія цього видання опублікована на сайті кафедри програмних систем і технологій за адресою: <https://pst.knu.ua/library>.

ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ

Освітня компонента «Програмні технології машинного навчання» входить до плану підготовки студентів спеціальності 121 «Інженерія програмного забезпечення» та є вибірковою у підготовці фахівців даного напрямку. Вона присвячена отриманню студентом компетенцій для того, щоб приймати участь у проектуванні програм обробки інформації та прогнозування, створення складних програм різних типів з використанням алгоритмів машинного навчання та інтегруванням у них програмних технологій, що їх реалізують.

Завдання вивчення освітньої компоненти полягає у формуванні у студентів знань про основні методи машинного навчання, їх математичну основу та програмні технології, що їх реалізують; термінологію та основні поняття з галузі машинного навчання, його основні проблеми і завдання; навчання з вчителем та навчання без вчителя; підготовку даних, їх передобробку та вибір методу; лінійну, поліноміальну та логістичну регресії і роботу з ними; тренування, візуалізацію та передбачення за допомогою дерев та лісів; архітектуру та навчання глибоких нейронних мереж та роботу з ними у Keras.

Кожна лабораторна робота виконується студентом під керівництвом викладача та включає наступні етапи:

- вивчення відповідного теоретичного матеріалу лекцій, лабораторних робіт та завдання;
- виконання завдання на комп'ютері з використанням відповідних програмних засобів та аналіз отриманих результатів.

Оцінювання знань та навиків, які здобув студент при виконання лабораторної роботи, проводиться викладачем під час захисту роботи.

Зміст

Лабораторна робота 1. Основи обробки числових даних.....	5
Лабораторна робота 2. Основи обробки категоріальних та текстових даних.....	8
Лабораторна робота 3. Прогнозування за допомогою лінійної та поліноміальної регресійних моделей	11
Лабораторна робота 4. Прогнозування за допомогою регуляризованих регресійних моделей.....	13
Лабораторна робота 5. Класифікація за допомогою методу опорних векторів.....	17
Лабораторна робота 6. Класифікація та регресія на основі дерев прийняття рішень	22
Лабораторна робота 7. Класифікація на основі ансамблевих методів...	25
Лабораторна робота 8. Метод головних компонент як зменшення розмірності даних	27
Лабораторна робота 9. Класифікація багатошаровим персептроном.....	29
Лабораторна робота 10. Класифікація на основі згорткових нейронних мереж	33
Додатки.....	40
Список літератури	42

Лабораторна робота 1

ОСНОВИ ОБРОБКИ ЧИСЛОВИХ ДАНИХ

Мета – набути практичних навичок обробки, аналізу та візуалізації даних, як етапу передобробки у задачах машинного навчання

Теоретичні відомості

1. *Огляд бібліотек мови програмування Python для роботи з даними:*

NumPy. Бібліотека з відкритим вихідним кодом, яка має наступні можливості: підтримка багатовимірних масивів (включаючи матриці) та високорівневих математичних функцій для роботи з ними.

Pandas. Бібліотека робота з якою будується поверх бібліотеки NumPy, що є інструментом нижчого рівня. Вона надає спеціальні структури даних та операції для перетворень числових таблиць. Основні можливості бібліотеки: об'єкт `DataFrame` для перетворень індексованих масивів двовимірних даних, інструменти для обміну даними між структурами в пам'яті і файлами різних форматів, засоби поєднання даних і способи обробки відсутньої інформації, переформатування наборів даних, в тому числі створення зведених таблиць, зріз даних за значеннями індексу, розширені можливості індексування, вибірка з великих наборів даних, вставка і видалення стовпців даних, злиття і об'єднання наборів даних, ієрархічне індексування, тощо.

Scikit-Learn. Python-бібліотека, яка містить велику кількість алгоритмів для задач, пов'язаних з класифікацією і машинним навчанням в цілому. Вона спеціалізується на алгоритмах машинного навчання для вирішення завдань навчання з вчителем: класифікації (прогноз ознаки, безліч допустимих значень якого обмежена) і регресії (прогноз ознаки зі значеннями), а також для задач навчання без учителя: кластеризації (розбиття даних по класах, які модель визначить сама), зниження розмірності (подання даних в просторі меншої розмірності з мінімальними втратами корисної інформації) і детектування аномалій, тощо.

2. *Завантаження даних, створення пробного датасету та візуалізація даних:*

Для початку створимо робочу область у середовищі Jupyter Notebook, імпортуємо необхідні бібліотеки та завантажмо дані для опрацювання за посиланням <https://www.kaggle.com/data/188528>. Датасет «Ігри» представляє собою таблицю зі статистикою продажів комп'ютерних ігор та містить 16719 рядків на 16 колонок, серед яких: назва гри, платформа, дата випуску, жанр гри, виробник, обсяг продажів, обсяг продажів у Європі, обсяг продажів у Японії, обсяг інших продажів, глобальний обсяг продажів, оцінка критиків, тощо.

Наведемо код, який реалізує вказані вище кроки:

```
import pandas as pd
import numpy as np
```

```
df = pd.read_csv('file_name.csv')
```

Дана таблиця має наступний вигляд (Рис. 1):

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_Score	Critic_Count
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.96	3.77	8.45	82.53	76.0	51.0
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24	NaN	NaN
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.76	3.79	3.29	35.52	82.0	73.0
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.93	3.28	2.95	32.77	80.0	73.0
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00	0.00	0.01	0.00	0.01	NaN	NaN
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00	0.01	0.00	0.00	0.01	NaN	NaN
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Factory	0.00	0.00	0.01	0.00	0.01	NaN	NaN
16717	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01	0.00	0.00	0.00	0.01	NaN	NaN
16718	Winning Post 8 2016	PSV	2016.0	Simulation	Tecmo Koei	0.00	0.00	0.01	0.00	0.01	NaN	NaN

16719 rows x 16 columns

Рис 1. Скрін шматка датасету «Ігри» у робочій області Jupyter Notebook

Розглянемо пробний набір даних як 20% від загальної кількості рядків, а саме 3343 рядки. Після чого виконаємо серію експериментів з візуалізації. Перше – побудуємо відношення: Platform до Year_of_Release, Global_Sales до Year_of_Release, Global_Sales до Genre, Global_Sales до EU_Sales. Якщо аналізувати дві властивості з набору даних, то можна визначити кореляцію між ними. Для визначення кореляції будемо використовувати функцію `np.corrcoef()`. Вона повертає матрицю кореляції, яка є двовимірним масивом з коефіцієнтами кореляції (Рис.2.). Чим вище коефіцієнт – тим сильніша кореляція. А також побудуємо матрицю розсіювання (Рис.3.), яка використовується для оцінки коваріаційної матриці, наприклад, багатовимірного нормального розподілу за допомогою `scatter_matrix()`.

Наведемо приклад коду:

```
myData = df.iloc [:3343]
#діаграма відношень Global_Sales до EU_Sales
myData.plot (kind="scatter" x="Global_Sales", y=" EU_Sales")

#кореляція між Global_Sales до EU_Sales
x = myData ['Global_Sales']
y= myData ['EU_Sales']
r = np.corrcoef(x,y)

#коваріаційна матриця
df1 = df[['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']].iloc [:3343]
pd.plotting.scatter_matrix (df1, alfa=0.2)
```

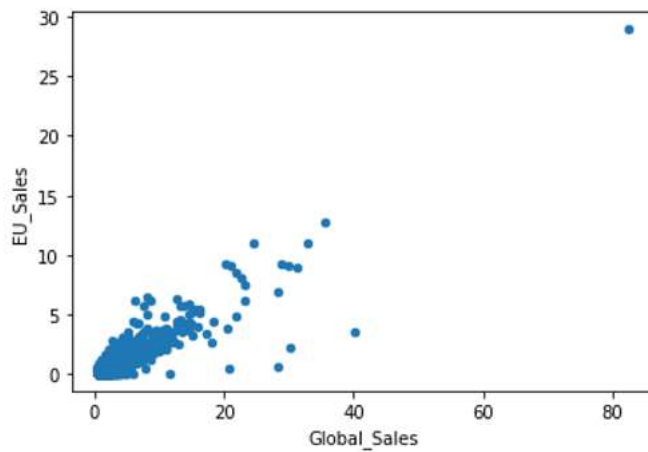


Рис 2. Діаграма відношення значень Global_Sales до EU_Sales

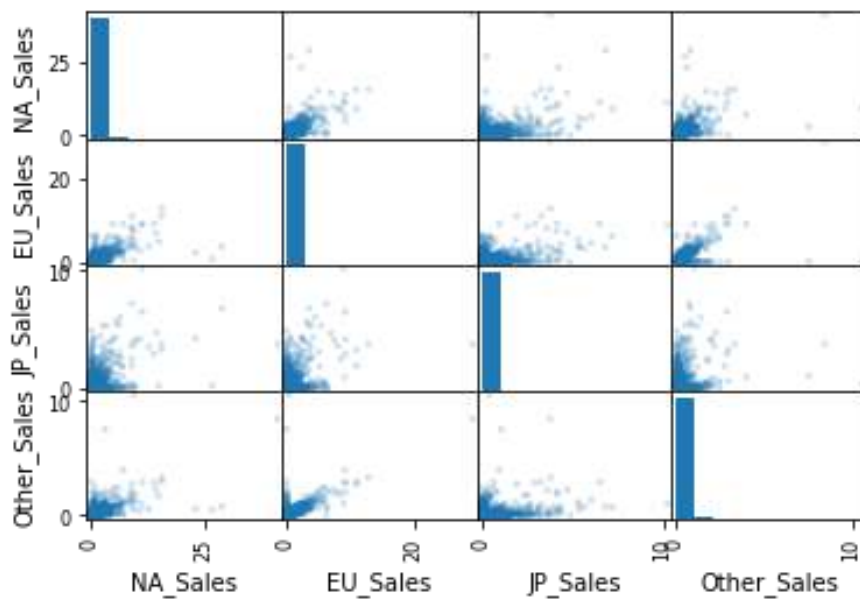


Рис 3. Коваріаційна матриця відношення значень 'NA_Sales', 'EU_Sales', 'JP_Sales' та 'Other_Sales'

3. Очищення, масштабування та стандартизація даних:

Очищення даних за допомогою видалення нульових та відсутніх даних, а також дублікатів. Реалізація цього відбувається за допомогою бібліотеки pandas і методів `dropnull(data)`, `dropna (data)` та `drop_duplicates(data)`.

Центрування та масштабування відбуваються шляхом обчислення відповідної статистики на зразках у навчальному наборі. Після чого середнє та стандартне відхилення зберігаються для використання у подальших даних із використанням перетворення. Стандартизація набору даних є загальною вимогою для багатьох оцінювачів машинного навчання: вони можуть давати погані результати, якщо окремі функції не виглядають як стандартні нормально розподілені дані. Альтернативним підходом до нормалізації (або стандартизації) є так зване масштабування Min-Max. При цьому дані масштабуються до

фіксованого діапазону - як правило, від 0 до 1. Наявність цього обмеженого діапазону - на відміну від стандартизації - полягає в тому, що ми в кінцевому підсумку отримаємо менші стандартні відхилення.

Наведемо приклад коду:

```
data = df [['NA_Sales', 'Global_Sales']].iloc [:3343]

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

MinMaxScaler.fit_transform (data)
StandardScaler.fit_transform (data)
```

Завдання

1. Обрати вхідні дані (датасет) для проведення обробки згідно варіанту (додаток 1), користуючись відкритими сховищами даних для машинного навчання.
2. Створити пробний набір даних (20 % від загальної кількості даних);
3. Візуалізувати дані пробного набору, побудувавши гістограми, графіки кореляцій за кожним з атрибутів та за їх комбінаціями;
4. Провести 5 серій по 4 експерименти, вибираючи різні пробні набори та реалізуючи завдання 3 для кожного з них. За результатами оформити звітну таблицю;
5. Очистити та масштабувати атрибути датасету.

Контрольні запитання

1. Що таке машинне навчання?
2. Які основні типи машинного навчання?
3. Які основні етапи машинного навчання?
4. Які основні етапи підготовки даних для задач машинного навчання?
5. Наведіть основні прийоми стандартизації даних?
6. Що таке очищення даних?

ЛІТЕРАТУРА: [1–3].

Лабораторна робота 2

ОСНОВИ ОБРОБКИ КАТЕГОРІАЛЬНИХ ТА ТЕКСТОВИХ ДАНИХ

Мета – набути навичок в обробці категоріальних та текстових даних з метою подальшого їх застосування у задачах машинного навчання.

Теоретичні відомості

1. *Обробка категоріальних даних:*

Один з варіантів кодування – Label Encoding, де кожному категорію замінюємо числом від 0 до $N - 1$, де N – кількість унікальних категорій. Однак в такому випадку категорії будуть мати логічний порядок розташування (категорія, якій відповідає більше число, буде мати більшу вагу). Для розглянутих

категоріальних характеристик авто таке кодування не підходить, оскільки дані категорії використовуються лише для ідентифікації.

Часто для кодування категоріальних атрибутів застосовують One-hot Encoding, в якому замінюємо кожну категорію одиничним вектором, що складається з нулів та одиниць, яка ідентифікує дану категорію (Рис.4). Для N унікальних категорій розмірність числового вектору буде відповідно N . Проте, зі зміною кількості ознак N так само буде змінюватись розмірність вектора, що не для всіх методів машинного навчання є прийнятним.

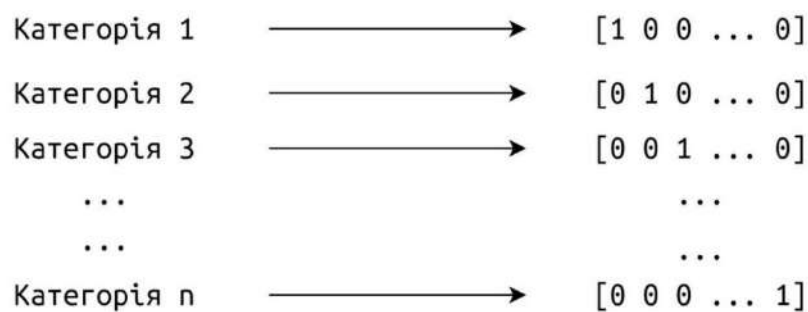


Рис. 4. Графічне представлення One-hot кодування

2. Обробка текстових даних:

Наведемо міркування, пов'язані з аналізом тексту п'єси «Measure for Measure» англійською мовою, які ґрунтуються на використанні спеціальної технології аналізу тексту, а саме за допомогою функціональності tidytext мовою R. Функції, методи тощо, пов'язані з цією мовою, написані під курсивом. Текст відтворення читався за допомогою спеціального файлу, тобто fileName, завантаження здійснювалося за допомогою методу readChar:

```
text <- readChar(fileName, file.info(fileName)$size). dChar(fileName,  
file.info(fileName)$size).
```

У результаті отримаємо вектор символів довжини, що дорівнює кількості прочитаних елементів. Далі перетворимо цей вектор символів у структуру таблиці даних, яка мовою R називається кадром даних:

```
text_df <- data_frame(text).
```

На цьому етапі весь текст розміщується в унікальній клітинці таблиці в стовпці, який називається текстом. Потім розбиваємо текст стовпця на токени (слова) за допомогою пакета tokenizers, розділяючи таблицю на одне слово на рядок:

```
tidy_text <- text_df %>% unnest_tokens(word, text).
```

Зауважимо, що функція unnest_tokens підтримує нестандартну оцінку через структуру tidyeval. Тут tidy_text — структура кадру даних, яка містить унікальне слово стовпця. Потім використовуємо пакет dplyr, що дозволяє виконувати запити на SQL для структур фрейму даних.

Наступний етап — видалення «стоп-слів». **Стоп-слова** - це загальноживані терміни, які не містять важливої інформації і, отже, не

мають значення у процесі розробки тексту. Англійські стоп-слова включають артиклі, іменники, розділові знаки, дієслова, прикметники, прислівники тощо. Пакет tidytext пропонує нам власні stop_words фрейму даних, що містять слово стовпець, наприклад, a, a's, able, about, above, тощо та лексикон, наприклад, SMART, який можна використовувати для видобутку тексту. Застосування функції anti_join:

```
tidy_text <- tidy_text %>% anti_join(stop_words)
```

отримуємо tidy_text, що містить лише слова, які мають значення. Частоту слів можна обчислити за допомогою функції count() з пакету dplyr:

```
tidy_text %>% count(word, sort = TRUE).
```

Отже, у tidy_text всі слова, які впорядковані за частотою, зазначеною в додатковому стовпці n. У випадку тексту «Measure for Measure» «15 найпопулярніших слів» представлено на рисунку 5.

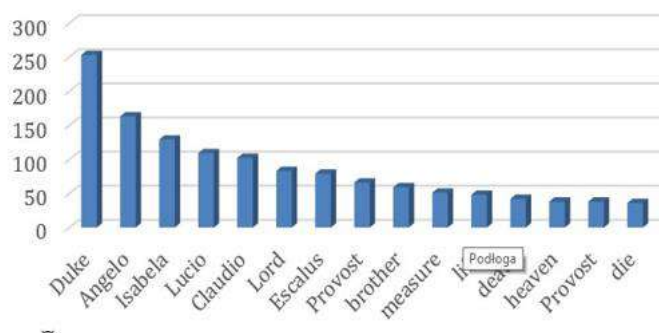


Рис. 5. Гістограма появи у тексті «Measure for Measure» 15 найпопулярніших слів

Після видалення стоп-слів кількість слів у документі все ще залишається дуже великою. Метод відбору ознак застосовується для подальшого зменшення розмірності, який базується на концепції використання функції оцінки, що застосовується до одного слова. Відомою є TF (term frequency) - відношення числа входжень вибраного слова до загальної кількості слів документа. Цей метод використовується для ранжирування всіх ознак, отриманих після попередньої обробки, а потім вибираються найвищі N оціночні характеристики на основі рейтингу. Аналогічні методи: mutual information, information gain, odds ratio, χ^2 statistic, term strength.

Наведемо формули для знаходження вектору ознак за методом tf-idf. Відзначимо, що це вектор числових значень, що може бути переданий у будь-який класифікатор. Спочатку обчислюємо tf :

$$tf = \begin{cases} 0, & \text{якщо слова } w_i \text{ немає в тексті } d, \\ \frac{n_{w_i}}{n}, & \text{якщо слово } w_i \in \text{в тексті } d, \end{cases}$$

де n_{w_i} – кількість входження слова w_i у текст d , а n – загальна кількість слів у тексті d .

Обчислюємо idf:

$$idf = \log \frac{N}{N_{w_i}},$$

де N – кількість документів, а N_{w_i} – кількість документів у текстах яких зустрічається слово w_i . Тоді загальна формула для коефіцієнту наступна: $tfidf = tf \times idf$.

Найбільше значення мають ті слова, що часто зустрічаються в рамках тексту d та рідко в інших документах колекції наявних документів. Знаходження tf - idf вектора реалізовано у `sklearn.feature_extraction.text.TfidfVectorizer()`.

Завдання

1. Обробити категоріальні дані датасету згідно варіанту (додаток 1).
2. Зібрати 20 текстових файлів від 1500 до 2000 символів кожен.
3. Провести обробку текстових файлів, виділивши числовий вектор ознак для кожного.

Контрольні запитання

1. У чому полягає процес обробки категоріальних даних для задач машинного навчання?
2. У чому полягає процес обробки текстових даних для задач машинного навчання?
3. У чому мета процесу видалення стоп-слів у текстах?

ЛІТЕРАТУРА: [1–4].

Лабораторна робота 3

ПРОГНОЗУВАННЯ ЗА ДОПОМОГОЮ ЛІНІЙНОЇ ТА ПОЛІНОМІАЛЬНОЇ РЕГРЕСІЙНИХ МОДЕЛЕЙ

Мета – набути навичок у реалізації задачі прогнозування за допомогою лінійної та поліноміальної регресійних моделей з різними алгоритмами пошуку параметрів.

Теоретичні відомості

1. Лінійна та поліноміальна регресійні моделі:

Регресія дозволяє за величиною однієї ознаки (змінна x) знаходити середні (очікувані) значення іншої ознаки (змінна y), зв'язаної з x кореляційно. Оскільки в дослідженнях конкретний вид взаємозв'язків невідомий, одне з головних завдань регресійного аналізу полягає у підборі відповідного виразу $y = f(x)$, графік якого проходить через емпіричні точки (або досить близько до них) і таким чином зв'язує змінні x та y .

Лінійна регресія має вигляд: $y = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n + u$, де y – прогнозоване значення, n – кількість ознак, β_i – i -ий параметр моделі, x_n – значення i -ої ознаки.

Задачею є знаходження коефіцієнтів регресії β_i . Серед програмних технологій, що реалізують дану задачу буде використовуватись метод `LinearRegression` бібліотеки `sklearn.linear_model`, який визначає коефіцієнти регресії методом найменших квадратів.

Поліноміальна регресія - це форма регресійного аналізу, при якій зв'язок між незалежною змінною x та залежною змінною y моделюється як поліном n -го степеня по x . Поліноміальна регресія відповідає нелінійній залежності між значеннями x та відповідним умовним середнім значенням y .

Якщо степінь поліноміальної регресії рівний d та існує n ознак, то з'явиться $\frac{(n+d)!}{d!n!}$ ознак (при 3 степеневій поліноміальній регресії за двома ознаками a та b отримуємо $a^2, b^2, a^3, b^3, ba^2, ab^2, ab$).

Для побудови поліному використаємо метод `PolynomialFeatures` зі пакету `sklearn.preprocessing` та накладемо його до `LinearRegression`.

2. *Перевірка даних на можливість застосування регресійної моделі:*

Чи є похибка є випадковою величиною із нульовим умовним математичним сподіванням для описових змінних? Якщо ні, то спробуйте виконати заміну незалежних змінних за деяким законом $\ln(x_i)$ або $\sqrt[n]{x_i}$.

Чи незалежні змінні виміряні без похибки? Якщо це не так, то спробуйте виконати заміну незалежних змінних за деяким законом $\ln(x_i)$ або $\sqrt[n]{x_i}$.

Чи є незалежні змінні лінійно незалежними? Якщо це не так, то корельовані змінні забирають з регресії. Критерій відбору - це значення коефіцієнту кореляції. Або заміна змінних в яких висока кореляція на їх лінійну комбінацію.

Чи є похибки не корельовані? Тобто, коваріаційна матриця похибок є діагональною, а кожен не нульовий елемент цієї матриці є дисперсією похибки.

Чи є дисперсія похибки сталою поміж спостережень (гомоскедастичність)? Якщо це не так, то спробуйте виконати заміну незалежних змінних за деяким законом $\ln(x_i)$, $\sqrt[n]{x_i}$ чи методом найменших квадратів.

3. *Програмна реалізація лінійної та поліноміальної регресійних моделей:*

Наведемо приклад програмної реалізації знаходження коефіцієнтів лінійної регресії та точності моделі:

```
from sklearn import linear_model

df = pd.read_csv('file_name.csv')
#вибір незалежної ознаки у column_i
x= df['column_i']
#вибір залежної ознаки у column_j
y = df['column_j']
#побудова тренувальної вибірки
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.1, random_state = 0)
#побудова лінійної регресії
reg = linear_model.LinearRegression()
reg.fit(x_train, y_train)
#вивід коефіцієнтів рівняння
reg.coef_
#вивід точності моделі
y_pred = reg.predict(x_test)
r2_score(y_test, y_pred)
```

Приклад програмної реалізації знаходження коефіцієнтів поліноміальної регресії степеню 2 :

```
from sklearn.preprocessing import PolynomialFeatures  
  
poly = PolynomialFeatures(degree=2)  
x= df[['column_i'], ['column_j']]  
poly.fit_transform(x)
```

Зауважимо, що лінійна і поліноміальна регресійні моделі є дуже популярними, проте в процесі дослідження складних практичних задач необхідно користуватись більш складнішими моделями з елементами регуляризації.

Завдання

1. Вивчити лінійну та поліноміальну регресійні моделі;
2. Перевірити дані датасету згідно варіанту (додаток 1) з метою можливості застосування регресійної моделі;
3. Побудувати лінійну та поліноміальну регресійні моделі за даними згідно варіанту, забезпечивши процес знаходження параметрів за допомогою аналітичного методу та градієнтним спуском;
4. Проаналізувати результати отримані у завданні 3.

Контрольні запитання

1. Що таке лінійна регресійна модель?
2. Які алгоритми пошуку розв'язків лінійної регресійної моделі існують?
3. Що таке поліноміальна регресійна модель?
4. Що таке аналітичний метод знаходження параметрів регресійної моделі?
5. Що таке градієнтний метод знаходження параметрів регресійної моделі?

ЛІТЕРАТУРА: [1–3, 6-10].

Лабораторна робота 4

ПРОГНОЗУВАННЯ ЗА ДОПОМОГОЮ РЕГУЛЯРИЗОВАНИХ РЕГРЕСІЙНИХ МОДЕЛЕЙ

Мета – набути практичних навичок у реалізації задачі прогнозування за допомогою регуляризованих регресійних моделей з різними алгоритмами пошуку параметрів

Теоретичні відомості

1. *Регуляризовані регресійні моделі:*

Регуляризація — загальний метод, що полягає в накладенні додаткових обмежень на шукані параметри, які можуть запобігти появі зайвої складності моделі. Сенс процедури полягає в "стягуванні" в ході налаштування вектора

коефіцієнтів таким чином, щоб вони в середньому виявилися дещо менше за абсолютною величиною, ніж це було б при оптимізації за методом найменших квадратів.

Гребнева регресія (регуляризація Тихонова) — це метод аналізу даних множинної регресії, які страждають від мультиколінеарності. Коли присутня мультиколінеарність, оцінки найменших квадратів є неупередженими, але їхні відхилення великі, тому вони можуть давати високу похибку у прогнозуванні значень. Додаючи ступінь упередженості до оцінок регресії, гребнева регресія зменшує стандартні похибки. Для розв'язку задач регресії за даним методом можна використовувати метод Ridge з пакету `sklearn.linear_model`.

Метод *регресії "ласо"* (LASSO, Least Absolute Shrinkage and Selection Operator) полягає у введенні додаткової складової регуляризації в функціонал оптимізації моделі, що часто дозволяє отримувати більш стійке рішення. Умова мінімізації квадратів помилки при оцінці параметрів β виражається наступною формулою: $\hat{\beta} = \arg \min(\sum_{i=1}^n (y_i - \sum_{j=1}^m \beta_j x_{ij})^2 + \lambda |\beta|)$, де λ - параметр регуляризації, який має означає штраф за складність.

Логістична регресія — статистичний регресійний метод, що застосовують у випадку, коли залежна змінна є категорійною, тобто може набувати тільки двох значень. Якщо оцінка імовірності, що зразок належить першому класу більша за 0.5, то рішення про належність – позитивне (1), в протилежному випадку ні (0).

Описується за формулою:

$$p = \frac{1}{1 + e^{-(\theta^T X)}}$$

де $\frac{1}{1 + e^{-(\theta^T X)}}$ - сигмоїдальна функція.

2. Програмна реалізація регуляризованих регресійних моделей:

Аналогічно до випадку лінійної моделі, регуляризовані реалізовані у вигляді готових рішень у `scikit-learn`. Для використання ласо-регресії необхідно використати `sklearn.linear_model.Lasso` (`alpha=1.0`, `*`, `fit_intercept=True`, `precompute=False`, `copy_X=True`, `max_iter=1000`, `tol=0.0001`, `warm_start=False`, `positive=False`, `random_state=None`, `selection='cyclic'`).

Простий приклад коду для реалізації ласо-регресії на множині незалежних змінних X та залежних Y має наступний вигляд:

```
from sklearn import linear_model

clf = linear_model.Lasso(alpha=0.1)
clf.fit(X, Y)
Lasso(alpha=0.1)
print(clf.coef_)
print(clf.intercept_)
```

Відмітимо, що реалізація вибору ласо-регресійної моделі існує за декількома критеріями: перехресної перевірки (`LassoCV`) та інформаційних критеріїв Акаїке (AIC) та Байєса (BIC). В останньому знаходження оптимального значення α є обчислювально дешевшою альтернативою, оскільки шлях

регуляризації обчислюється лише один раз замість $k + 1$ разів при використанні k -кратної перехресної перевірки.

Реалізуємо лассо-регресію з використанням алгоритму LARS на наборі даних про захворювання діабетом. Кожен колір представляє різні ознаки вектора коефіцієнта, і це відображається як функція параметра регуляризації. Приклад програмного коду на мові Python у середовищі Jupiter notebook наведено на Рис. 6.

Аналогічно розглянемо логістичну регресію. Згенеруємо штучний набір даних у вигляді лінійної залежності з деяким гаусовим шумом (Рис.8).

```
In [1]: print(__doc__)

import numpy as np
import matplotlib.pyplot as plt

from sklearn import linear_model
from sklearn import datasets

X, y = datasets.load_diabetes(return_X_y=True)

print("Computing regularization path using the LARS ...")
_, _, coefs = linear_model.lars_path(X, y, method='lasso', verbose=True)

xx = np.sum(np.abs(coefs.T), axis=1)
xx /= xx[-1]

plt.plot(xx, coefs.T)
ymin, ymax = plt.ylim()
plt.vlines(xx, ymin, ymax, linestyle='dashed')
plt.xlabel('|coef| / max|coef|')
plt.ylabel('Coefficients')
plt.title('LASSO Path')
plt.axis('tight')
plt.show()
```

Рис. 6. Скрін реалізації алгоритму LARS на наборі даних про захворювання діабетом

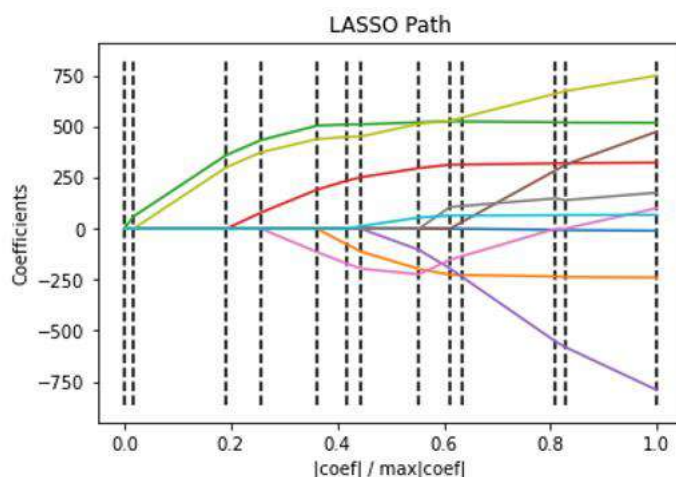


Рис. 7. Графік залежності між коефіцієнтами алгоритму LARS на наборі даних про захворювання діабетом

```
xmin, xmax = -5, 5
n_samples = 100
np.random.seed(0)
X = np.random.normal(size=n_samples)
y = (X > 0).astype(np.float)
X[X > 0] *= 4
X += .3 * np.random.normal(size=n_samples)
X = X[:, np.newaxis]
```

Рис. 8. Скрін генерації набору даних

Для реалізації скористаємось `sklearn.linear_model.LogisticRegression` (penalty='l2',*, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import linear_model
from scipy.special import expit

# встановлюємо класифікатор
clf = linear_model.LogisticRegression(C=1e5)
clf.fit(X, y)

# і побудуємо результат
plt.figure(1, figsize=(4, 3))
plt.clf()
plt.scatter(X.ravel(), y, color='black', zorder=20)
X_test = np.linspace(-5, 10, 300)

loss = expit(X_test * clf.coef_ + clf.intercept_).ravel()
plt.plot(X_test, loss, color='red', linewidth=3)

ols = linear_model.LinearRegression()
ols.fit(X, y)
plt.plot(X_test, ols.coef_ * X_test + ols.intercept_, linewidth=1)
plt.axhline(.5, color='.5')

plt.ylabel('y')
plt.xlabel('X')
plt.xticks(range(-5, 10))
plt.yticks([0, 0.5, 1])
plt.ylim(-.25, 1.25)
plt.xlim(-4, 10)
plt.legend(('Logistic Regression Model', 'Linear Regression Model'),
          loc="lower right", fontsize='small')
plt.tight_layout()
plt.show()
```

Рис. 9. Скрін реалізації логістичної регресії на штучно згенерованому наборі даних

Завдання

1. Вивчити гребневу, лассо- та логістичну регресійні моделі;
2. Перевірити дані датасету згідно варіанту (додаток 1) з метою можливості застосування регресійної моделі;
3. Побудувати гребневу, лассо- та логістичну регресійні моделі за даними згідно варіанту;
4. Проаналізувати результати отримані у завданні 3.

Контрольні запитання

1. Що таке гребнева регресія?
2. Що таке лассо-регресія?
3. Що таке логістична регресія?

ЛІТЕРАТУРА: [1–3, 6-10].

Лабораторна робота 5

КЛАСИФІКАЦІЯ ЗА ДОПОМОГОЮ МЕТОДУ ОПОРНИХ ВЕКТОРІВ

Мета – набутти практичних навичок у реалізації задачі класифікації за допомогою методу опорних векторів з жорстким та м'яким проміжками.

Теоретичні відомості

1. Класифікація та її оцінка:

Нехай X -множина даних, а Y -скінченне число класів, що помічені. Існує відображення $f: X_1 \rightarrow Y$, яке відоме на навчаючих даних $X_1 \subset X$. Необхідно отримати алгоритм $a: X \rightarrow Y$, який класифікує будь-який x з множини X і $f(X_1) = a(X_1)$, де Y – дискретна множина.

Двійкові класифікатори (binary classifier) дають градацію «щось» або «не щось». Їх часто називають лінійними.

Оцінюємо двійковий класифікатор за наступними критеріями:

- Перехресна перевірка: робимо репрезентативну вибірку пропорційно з кожного класу, ітеративно робимо клон класифікатора, навчаємо його на навчаючих даних, перевіряємо роботу на тестуючих даних та підраховуємо відсоток правильних відповідей;

- Матриця неточностей для двійкового класифікатора має вигляд: TN (true negative)- коректно класифіковані як «не щось»; FP (false positive) – хибно класифіковані як «щось»; FN (false negative) – хибно класифіковані як «не щось»; TP (true positive) – істинно класифіковані як «щось»; TN та FP утворюють негативний клас NC (negative class); PC та FN утворюють позитивний клас PC (positive class);

- Крива ROC зображає частку істинно позитивних класифікацій по відношенню до частки хибно позитивних класифікацій;

- Площа кривої ROC.

Ознаки «хорошого» класифікатора:

- У матриці неточностей $FP = 0$ і $FN = 0$;
- Площа під кривою ROC рівна 1;
- Крива ROC максимально притискається до лівого верхнього кута.

2. Метод опорних векторів:

Прогнозує клас нового зразка x , обчислюючи функцію розв'язку

$$w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b = wx + b :$$

$$\hat{y} = \begin{cases} 0, & \text{якщо } wx + b < 0, \\ 1, & \text{якщо } wx + b \geq 0. \end{cases}$$

Невідомими є w та b .

Для їх обчислення з метою **жорсткого проміжку** розв'яжемо задачу умовної оптимізації:

$$\min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \mathbf{w} \text{ при умові } t^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1, i = \overline{1, m}.$$

Для їх обчислення з метою **м'якого проміжку** вводимо фіктивну змінну $\xi^{(i)}$, яка визначає на скільки i -ий зразок може порушувати зазір та гіперпараметр C , який вирішує компроміс між двома суперечливими задачами (мінімізувати вираз, щоб розширити проміжок та зменшувати $\xi^{(i)}$ для меншого втручання у смугу) розв'яжемо задачу умовної оптимізації:

$$\min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^m \xi^{(i)} \text{ при умові } t^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1 - \xi^{(i)}, \xi^{(i)} \geq 0, i = \overline{1, m}.$$

Основна ідея методу полягає в побудові у n -вимірному просторі ознак гіперплощини розмірності $(n - 1)$, яка розділяє простір на два шматки, що відповідають двом класам.

3. Програмна реалізація методу опорних векторів:

Для дослідження методу опорних векторів (SVM) згенеруємо датасет за допомогою методу `make_blobs` from `sklearn.datasets`. Нехай кількість зразків дорівнює 200, а тестовий набір складає 20% від загального.

Код:

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

```
X, y = make_blobs(n_samples=120, centers=2, random_state=0, cluster_std=0.70)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

x

```
array([[ 2.59193175,  1.14706863],
       [ 1.7756532 ,  1.15670278],
       [ 2.8032241 ,  0.5802936 ],
       [ 1.6090616 ,  0.61957339],
       [ 2.04921553,  5.33233847],
       [ 0.50554777,  4.05210011],
       [ 1.07710058,  5.32177878],
       [ 0.35482006,  2.9172298 ],
       [ 1.96225112,  0.68921004],
       [-0.16486876,  4.62773491],
       [ 1.63784769,  0.86086655],
       [ 0.97670961,  0.94194706],
       [ 1.23976795,  3.53420677],
       [ 0.09931333,  4.84803058],
       [ 2.123675 ,  1.30573124],
       [ 0.41388331,  3.82110248],
       [ 0.47637256,  0.18190887],
       ...])
```

Рис 10. Скрін згенерованого датасету

Застосуємо лінійний метод опорних векторів SVM з жорстким проміжком, використовуючи SVC from sklearn.svm та візуалізуємо проміжки:

```
def plot_svc (model, ax=None):
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # генеруємо решітку
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # графік розв'язку та межі
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    ax.scatter(model.support_vectors_[0],
               model.support_vectors_[1],
               s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

model = SVC(kernel='linear', C=10)
model.fit(X_train, y_train)
plot_svc (model, ax)
```

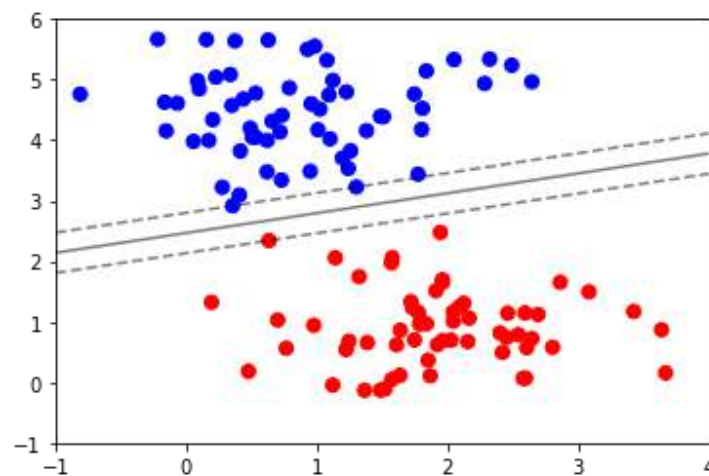


Рис 11. Візуалізація SVM з жорстким проміжком для згенерованого датасету
Зауважимо, що суцільна пряма на Рис. 11 – максимізує відступ між двома множинами точок. Декілька тренувальних точок, що доторкаються до прямої – є

основними елементами для побудови моделі – опорними векторами. Їх координати зберігаються у моделі у полі `support_vectors_`.

Спрогнозуємо у для `X_test` та оцінимо точність моделі за допомогою матриці неточностей та методу `classification_report` from `sklearn.metrics`.

```
y_pred = model.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

Результат: $\begin{bmatrix} 23 & 0 \\ 0 & 17 \end{bmatrix}$

Оцінки класифікатора мають наступний вигляд:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	23
1	1.00	1.00	1.00	17
accuracy			1.00	40
macro avg	1.00	1.00	1.00	40
weighted avg	1.00	1.00	1.00	40

Пояснимо терміни: `recall` – кількість зразків, що було класифіковано правильно ($\text{recall} = \text{TP} / (\text{TP} + \text{FN})$); `precision` (точність) – кількість зразків, які було передбачено правильно з усіх позитивних сценаріїв ($\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$); `F-Score` – гармонійне середнє значення `Precision` та `Recall`; $\text{F-Score} = (2 * \text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$.

Додатково побудуємо криву ROC:

```
def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

plot_roc_curve(fpr, tpr)

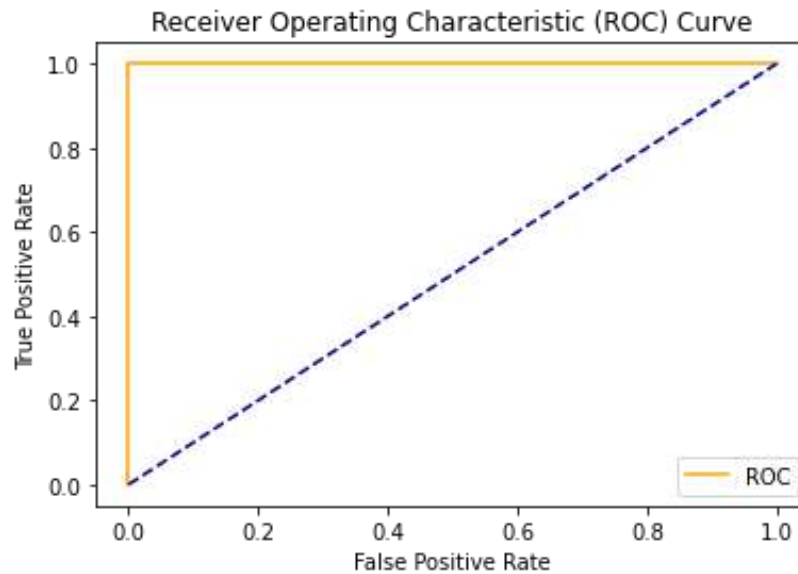


Рис 12. Крива ROC

Реалізуємо цей же метод SVM з м'яким проміжком. Для цього згенеруємо трохи «неідеальний» датасет, де класи будуть частково змішані.

Код:

```
X, y = make_blobs(n_samples=200, centers=2,  
                  random_state=0, cluster_std=1.2)
```

За регуляцію жорсткості проміжку моделі відповідає параметр C . Він дозволяє певним точкам проникати в поле відступу. Чим більше значення параметру, тим жорсткіший проміжок. Для минулого прикладу значення C дорівнювало 10. В цьому прикладі ми встановимо його у 0.1.

Оцінимо точність моделі спрогнозувавши значення для X_{test} та обчисливши попередні метрики.

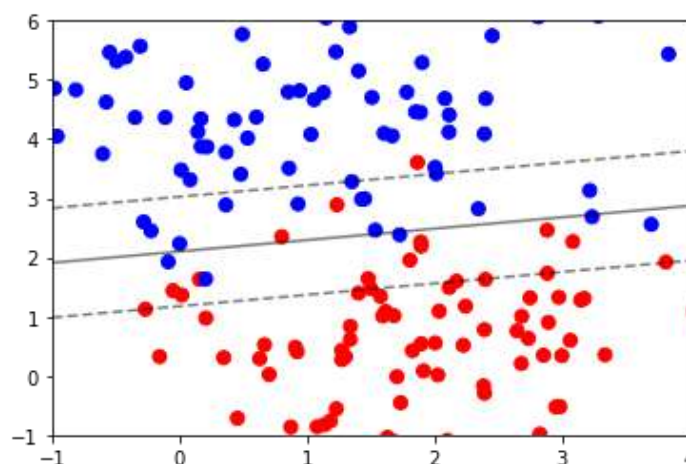


Рис 13. Візуалізація SVM з м'яким проміжком для згенерованого датасету

Матриця неточностей: $\begin{bmatrix} 23 & 0 \\ 3 & 14 \end{bmatrix}$

Оцінки класифікатора мають наступний вигляд:

	precision	recall	f1-score	support
0	0.88	1.00	0.94	23
1	1.00	0.82	0.90	17
accuracy			0.93	40
macro avg	0.94	0.91	0.92	40
weighted avg	0.93	0.93	0.92	40

У даному випадку 3 зразки були класифіковані як false positive, через що модель отримала точність 93%.

Завдання

1. Вивчити модель лінійного класифікатора SVM з жорстким та м'яким проміжками;
2. Реалізувати програмно модель лінійного класифікатора SVM з жорстким та м'яким проміжками відповідно до даних згідно варіанту (додаток 1);
3. Оцінити їх роботу за допомогою перехресної перевірки, матриці неточностей, кривої ROC та площею під нею;
4. Проаналізувати отримані результати.

Контрольні запитання

1. Які типи класифікаторів на основі методу опорних векторів існують?
2. Які є методики розв'язання нелінійних задач методу опорних векторів?
3. Що таке динамічний метод опорних векторів?

ЛІТЕРАТУРА: [1-3, 6-10].

Лабораторна робота 6

КЛАСИФІКАЦІЯ ТА РЕГРЕСІЯ НА ОСНОВІ ДЕРЕВ ПРИЙНЯТТЯ РІШЕНЬ

Мета – набути навичок у побудові дерев рішень з метою реалізації задач класифікації та регресії, користуючись різними гіперпараметрами регуляції.

Теоретичні відомості

Дерева прийняття рішень (decision tree) – універсальний апарат, який розв'язує всі задачі машинного навчання на будь-яких наборах даних. Їх називають моделлю білого ящика – рішення легко інтерпретувати людині. У протипагу нейронним мережам чи випадковим лісам (моделі чорного ящика).

CART. В цьому алгоритмі кожен вузол дерева рішень має двох нащадків. На кожному кроці побудови дерева у вузлі формулюється правило, яке ділить навчальну вибірку на дві частини, – частина, в якій виконується правило (нащадок - right), і частина, в якій правило не виконується (нащадок - left). Для вибору оптимального правила використовується функція оцінки якості розбиття.

Механізм відсікання дерева - найбільш серйозна відмінність алгоритму CART від інших алгоритмів побудови дерева. CART розглядає відсікання як

отримання компромісу між двома проблемами: отримання дерева оптимального розміру і отримання точної оцінки ймовірності помилкової класифікації.

Навчаючий набір розділяємо на два піднабори за однією з ознак k та порогом t_k . Для цього вибирають найчистішу пару (k, t_k) . Далі, ітеративно розділяємо на два кожен отриманий піднабір (отримуємо двійкове дерево). Зупиняємо: або у випадку досягнення параметру глибини `max_depth` або не скорочується забруднення.

Функція втрат $J(k, t_k) = \frac{m_{\text{лівий}}}{m} G_{\text{лівий}} + \frac{m_{\text{правий}}}{m} G_{\text{правий}}$, де $m_{\text{лівий/правий}}$ - кількість зразків у лівому/правому піднаборі; $G_{\text{лівий}}$ - забрудненість лівого/правого піднабору.

Гіперпараметри регуляції побудови дерева:

1. Обмеження глибини дерева;
2. Мінімальна кількість зразків, які мають бути в класі перед тим як його розділити (`min_samples_split`);
3. Мінімальна кількість зразків у листовому вузлі (`min_samples_leaf`);
4. Мінімальна частка від загальної кількості зважених зразків (`min_weight_fraction_leaf`);
5. Максимальна кількість листових вузлів (`max_leaf_nodes`);
6. Максимальна кількість ознак, які оцінюються при розділенні вузла (`max_features`).

Оцінка якості побудованого дерева рішень:

Неоднорідність Джинні: $G_i = 1 - \sum_{k=1}^n p_{i,k}^2$, де $p_{i,k}$ - доля зразків класу k серед навчаючих зразків у i -ому вузлі.

Ентропія: $H_i = - \sum_{p_{i,k} \neq 0} p_{i,k} \log(p_{i,k})$.

2. Програмна реалізація дерев рішень з метою задач класифікації та регресії:

Наведемо приклад реалізації задачі класифікації.

Код:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
from sklearn import metrics
import pydotplus
import numpy as np
import pandas as pd
import csv
```

```
X = pd.read_csv('data.csv', header=None)# завантажуюємо атрибути
```

```
Y = pd.read_csv('label.csv', header=None)# завантажуюємо класи
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.1, random_state = 0)
```

```

clf = DecisionTreeClassifier(max_depth=5) # класифікатор з обмеженням глибини
clf = clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)

result1 = classification_report(y_test, y_pred)
print("Classification Report:")
print(result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data, filled=True, rounded=True,
    special_characters=True,feature_names = ['f1','f2','f3'],class_names=['no','yes']) #
'f1','f2','f3' відповідає кількості стовбців у файлі data.csv

# побудова дерева у файл Tree_binary.png
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('Tree_binary.png')
Image(graph.create_png())

```

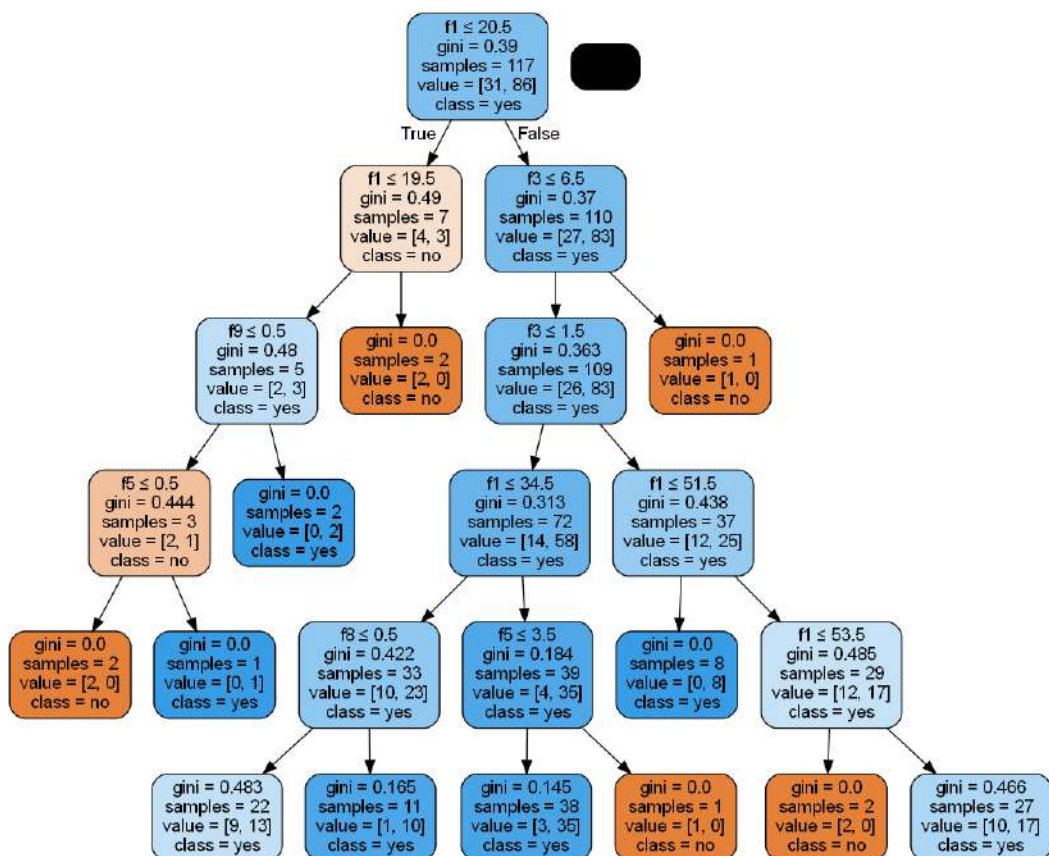


Рис 14. Дерево прийняття рішень з обмеженням глибини у 5

Наведемо приклад реалізації задачі регресії.

Код:

```

from sklearn.datasets import load_diabetes
from sklearn.model_selection import cross_val_score

```

```

from sklearn.tree import DecisionTreeRegressor

X = pd.read_csv('data.csv', header=None)# завантажуюмо атрибути
Y = pd.read_csv('label.csv', header=None)# завантажуюмо класи

regressor = DecisionTreeRegressor(random_state=0)
cross_val_score(regressor, X, Y, cv=10)

```

Завдання

1. Вивчити основи побудови дерев рішень.
2. Реалізувати алгоритм побудови дерева рішень з різними гіперпараметрами регуляції для перенавчання з метою класифікації даних згідно варіанту (додаток 1);
3. Реалізувати алгоритм CART з різними гіперпараметрами регуляції для перенавчання з метою регресії.
4. Проаналізувати отримані результати.

Контрольні запитання

1. Що таке алгоритм CART для дерев рішень?
2. Як реалізується задача регресії для дерев рішень?
3. Які переваги та недоліки дерев рішень?

ЛІТЕРАТУРА: [1-3, 6-10].

Лабораторна робота 7

КЛАСИФІКАЦІЯ НА ОСНОВІ АНСАМБЛЕВИХ МЕТОДІВ

Мета – набути практичних навичок у реалізації задачі класифікації даних на основі одного з ансамблевих методів (беггінгу, бустингу або стекінгу).

Теоретичні відомості

1. *Ансамблеві методи класифікації:*

Ансамблеві методи - це парадигма машинного навчання, де кілька моделей (часто званих «слабкими учнями») навчаються для вирішення однієї і тієї ж проблеми і об'єднуються для отримання кращих результатів. Основна гіпотеза полягає в тому, що при правильному поєднанні слабких моделей ми можемо отримати більш точні і / або надійні моделі.

Випадковий ліс — ансамблевий метод машинного навчання для класифікації, регресії та інших завдань, які оперують за допомогою побудови численних дерев прийняття рішень під час тренування моделі і продукують моду для класів (класифікацій) або усереднений прогноз (регресія) побудованих дерев. В основі лежить алгоритм беггінгу. Недоліком є схильність до перенавчання.

Беггінг. Нехай навчальна вибірка складається з N прикладів, розмірність простору ознак дорівнює M , і заданий параметр $m \approx \sqrt{M}$.

Усі дерева будуються незалежно один від одного за такою процедурою:

1. Згенеруємо випадкову підвибірку з **повторенням** розміром n з навчальної вибірки.

2. Побудуємо дерево рішень, яке класифікує дану підвибірку, причому в ході створення чергового вузла дерева будемо вибирати ознаку, на основі якої проводиться розбиття, не з усіх M ознак, а лише з m випадково вибраних. Вибір найкращої з цих m ознак може здійснюватися різними способами. В оригінальному коді Брейман використовується критерій Джині, що застосовується також в алгоритмі побудови вирішальних дерев CART.

3. Дерево будується до повного вичерпання підвибірки і не піддається процедурі відсікання.

Класифікація об'єктів проводиться шляхом голосування: кожне дерево комітету відносить об'єкт, який класифікується до одного з класів, і перемагає клас, за який проголосувало найбільше число дерев.

Оптимальне число дерев підбирається таким чином, щоб мінімізувати помилку класифікатора на тестовій вибірці. У разі її відсутності, мінімізується оцінка помилки out-of-bag: частка прикладів навчальної вибірки, неправильно класифікованих комітетом, якщо не враховувати голоси дерев на прикладах, що входять в їх власну навчальну підвибірку.

Серед недоліків відмітимо, що алгоритм схильний до перенавчання на деяких завданнях, особливо з великою кількістю шумів та великий розмір отримуваних моделей. Потрібно $O(NK)$ пам'яті для зберігання моделі, де K — число дерев.

Бустинг. Процедура послідовної побудови композиції алгоритмів машинного навчання, коли кожен наступний алгоритм прагне компенсувати недоліки композиції всіх попередніх алгоритмів. Бустинг є жадібним алгоритмом побудови композиції алгоритмів. Він є одним з найбільш популярних методів машинного навчання, поряд з нейронними мережами і машинами опорних векторів. Основні причини - простота, універсальність, гнучкість (можливість побудови різних модифікацій), і, головне, висока узагальнююча здатність.

Типи бустингу:

- AdaBoost;
- Градієнтний бустинг;
- Стекінг.

Стекінг. Його основна ідея полягає у використанні базових класифікаторів для отримання прогнозів (метаознак) і використанні їх як ознак для деякого "узагальнюючого" алгоритму (метаалгоритму). Іншими словами, основною ідеєю стекінгу є перетворення вихідного простору ознак завдання в новий простір, точками якого є передбачення базових алгоритмів. Пропонується спочатку вибрати набір пар довільних підмножин з навчальної вибірки, потім для кожної пари навчити базові алгоритми на першій підмножині і передбачити ними цільову змінну для другої підмножини. Передбачені значення і стають об'єктами нового простору.

2. *Програмна реалізація ансамблевих методів класифікації:*

Реалізуємо

Код:

```
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import csv
```

```
X = pd.read_csv('data.csv', header=None)# завантажуюмо атрибути
Y = pd.read_csv('label.csv', header=None)# завантажуюмо класи
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.1, random_state = 0)
```

Завдання

1. Вивчити беггінг, бустинг та стекінг як основні прийоми ансамблю для лісу прийняття рішень;
2. Реалізувати беггінг, бустинг та стекінг для задачі класифікації на основі даних згідно варіанту (додаток 1);
3. Проаналізувати отримані результати.

Контрольні запитання

1. Що таке беггінг як ансамблевий метод випадкового лісу прийняття рішень?
2. Що таке бустинг як ансамблевий метод випадкового лісу прийняття рішень?
3. Що таке стекінг як ансамблевий метод випадкового лісу прийняття рішень?

ЛІТЕРАТУРА: [1-3, 6-10].

Лабораторна робота 8

МЕТОД ГОЛОВНИХ КОМПОНЕНТ ЯК ЗМЕНШЕННЯ РОЗМІРНОСТІ ДАНИХ

Мета – набути навичок у реалізації методу головних компонент як способу зменшення розмірності даних для машинного навчання.

Теоретичні відомості

1. Метод головних компонент:

Головні компоненти – це такі комбінації початкових ознак:

$$x'_l = \alpha_{x',x} x_l + \alpha_{x',y} y_l,$$

$$y'_l = \alpha_{y',x} x_l + \alpha_{y',y} y_l, l = \overline{1, N},$$

де $\alpha_{x',x}^2 + \alpha_{x',y}^2 = 1$; $\alpha_{y',x}^2 + \alpha_{y',y}^2 = 1$; $\alpha_{x',x}$, $\alpha_{x',y}$, $\alpha_{y',x}$, $\alpha_{y',y}$ – відповідно косинуси кутів між осями x' і x , x' і y , y' і x та y' і y .

Положення осі x' визначається умовою максимуму дисперсії вибірки $\Omega_{2,N}$ вздовж неї; осі y' – умовою її ортогональності до x' . Метод обчислення напрямків головних осей x' , y' ґрунтується на знаходженні власних чисел і

власних векторів дисперсійно-коваріаційної матриці $\hat{DC}\{\vec{\xi}\}$, центрованих відносно середнього даних, або ж кореляційної матриці $\hat{R}\{\vec{\xi}\}$, на разі, коли дані стандартизовані. В останньому випадку власний вектор з індексом x' являє собою набір коефіцієнтів

$$\vec{A}_{x'} = (\alpha_{x',x}, \alpha_{x',y}),$$

а відповідне йому власне число

$$\lambda_{x'} = 1 + |\hat{r}_{x,y}|$$

дорівнює дисперсії компоненти x' . Тоді як при використанні матриці $\hat{R}\{\vec{\xi}\}$ сума власних чисел дорівнює кількості ознак, то знайшовши відношення $\frac{\lambda_{x'}}{2}$, – одержують частку дисперсії вибірки, що відповідає x' -му напрямку. Те ж саме, відповідно, стосується і y' -го напрямку (тут маємо $\lambda_{y'} = 1 - |\hat{r}_{x,y}|$). У будь-якому разі, як для нормованих, так і для стандартизованих даних, якщо позначити дисперсії головних компонент $\sigma_{x'}^2$, $\sigma_{y'}^2$, виявляється, що

$$\sigma_x^2 + \sigma_y^2 = \sigma_{x'}^2 + \sigma_{y'}^2,$$

отже, початково закладена в спостереженнях сумарна варіабельність при переході до нових змінних не змінюється, а лише перерозподіляється. Окрім того, нові змінні (компоненти), на відміну від початкових ознак, набувають таку цінну властивість, як відсутність кореляції поміж собою.

2. Програмна реалізація МГК:

У бібліотеці `scikit-learn` мови програмування `python` існує декілька реалізованих рішень: МГК, інкрементний МГК (Incremental PCA), МГК з використанням рандомізованого відкидання сингулярного вектора (PCA using randomized SVD), розріджений МГК (SparsePCA) та його міні-пакетна реалізація (MiniBatchSparsePCA).

Наведемо перелік класів, які їх реалізують:

- МГК - `sklearn.decomposition.PCA` (`n_components=None`, `*`, `copy=True`, `whiten=False`, `svd_solver='auto'`, `tol=0.0`, `iterated_power='auto'`, `n_oversamples=10`, `power_iteration_normalizer='auto'`, `random_state=None`);

- інкрементний МГК - `sklearn.decomposition.IncrementalPCA` (`n_components=None`, `*`, `whiten=False`, `copy=True`, `batch_size=None`);

- МГК з рандомізованим відкиданням - `sklearn.decomposition.PCA` (`n_components=None`, `*`, `copy=True`, `whiten=False`, `svd_solver='randomized'`, `tol=0.0`, `iterated_power='auto'`, `n_oversamples=10`, `power_iteration_normalizer='auto'`, `random_state=None`);

- розріджений МГК `sklearn.decomposition.SparsePCA` (`n_components=None`, `*`, `alpha=1`, `ridge_alpha=0.01`, `max_iter=1000`, `tol=1e-08`, `method='lars'`, `n_jobs=None`, `U_init=None`, `V_init=None`, `verbose=False`, `random_state=None`);

• міні-пакетний МГК - `sklearn.decomposition.MinibatchSparsePCA`
(`n_components=None`, *, `alpha=1`, `ridge_alpha=0.01`, `n_iter='deprecated'`,
`max_iter=None`, `callback=None`, `batch_size=3`, `verbose=False`, `shuffle=True`, `n_jobs=None`,
`method='lars'`, `random_state=None`, `tol=0.001`, `max_no_improvement=10`)

Розглянемо найпростішу реалізацію, код якої наведений нижче:

```
from sklearn.decomposition import PCA
import numpy as np
import pandas as pd
import csv

df = pd.read_csv('data.csv', header=None) # завантажуюмо датасет
X = df.to_numpy()
pca = PCA(n_components=10) # знаходимо 10 головних ознак вектора датасету
pca.fit(X)
print(pca.explained_variance_ratio_)
print(pca.singular_values_)
print ('Mean vector: ', pca.mean_)
```

Завдання

1. Вивчити основи методу головних компонент.
2. Реалізувати МГК на датасеті згідно варіанту (додаток 1), підбираючи пороги та варіабельність;
3. Проаналізувати отримані результати.

Контрольні запитання

1. Які основні підходи до зменшення розмірності даних існують?
2. Що таке локально-лінійне вкладення, як метод зменшення розмірності даних на основі многовидів?
3. Які недоліки методу головних компонент існують?
4. Які є модифікації методу головних компонент?

ЛІТЕРАТУРА: [1-3, 6-10].

Лабораторна робота 9

КЛАСИФІКАЦІЯ БАГАТОШАРОВИМ ПЕРСЕПТРОНОМ

Мета – набути навичок у реалізації багатошарового персептрону, як класифікатора в задачах машинного навчання.

Теоретичні відомості

1. Багатошаровий персептрон (БШП):

Машинне навчання є частиною штучного інтелекту, в рамках якого ми надаємо машині дані, щоб вона могла вивчити зразок з даних і змогла передбачити рішення подібних майбутніх проблем. Нейронна мережа (NN) натхненна нейронною мережею людського мозку. Багатошаровий персептрон є

найбільш простим серед тих, що дають можливість реалізовувати практично значимі результати у задачах класифікації.

Архітектура БШП зображена на Рис.15. Кожен нейрон має свою активаційну функцію, яке перетворює сигнал, який прийшов йому на вхід.

Одними з найбільш популярних є наступні активаційні функції:

1. $\text{Sigmoid} = \frac{1}{1+e^{-Cz}}$ (сигмоїдальна);
2. $\text{Th} = \frac{e^{Cz} - e^{-Cz}}{e^{Cz} + e^{-Cz}}$ (тангенс гіперболічний);
3. $\text{ReLU} = \max(z, 0)$.

Мета навчання мережі – для деякої множини входів давати бажану множину виходів. Навчання – послідовне представлення вхідних векторів з одночасним підлаштуванням коефіцієнтів ваги у відповідності з певною процедурою.

Основні методи навчання БШП (підлаштування ваг):

1. Метод градієнтного спуску;
2. Quick Prop;
3. Метод rProp;
4. Метод спряжених градієнтів (NAG);
5. AdaGrad (Adaptive Gradient);
6. AdaDelta;
7. Adam

Гарною практикою є використання наступних стратегій регулювання БШП для запобігання перенавчанню:

○ **Full Batch** : обробляємо навчаючий набір; визначаємо похибки; коректуємо параметри; перевіряємо на контрольній множині; якщо результат задовільний, то виконуємо заключний тест, в протилежному випадку переходимо на початок.

○ **Mini Batch** : випадковим чином вибираємо підмножину з навчаючого набору; визначаємо похибки на підмножині; коректуємо параметри; перевіряємо на контрольній множині; якщо результат задовільний, то виконуємо заключний тест, у протилежному випадку переходимо на початок.

○ **Stochastic** : випадковим чином вибираємо один приклад з навчаючого набору; обробляємо його; визначаємо похибку; зберігаємо даний стан параметрів; визначаємо нові параметри; перевіряємо на контрольній множині; якщо результат задовільний, то виконуємо заключний тест; якщо похибка виросла, то ймовірно, що пропорційна до зростання похибки виконуємо повернення до попереднього стану параметрів і переходимо на початок.

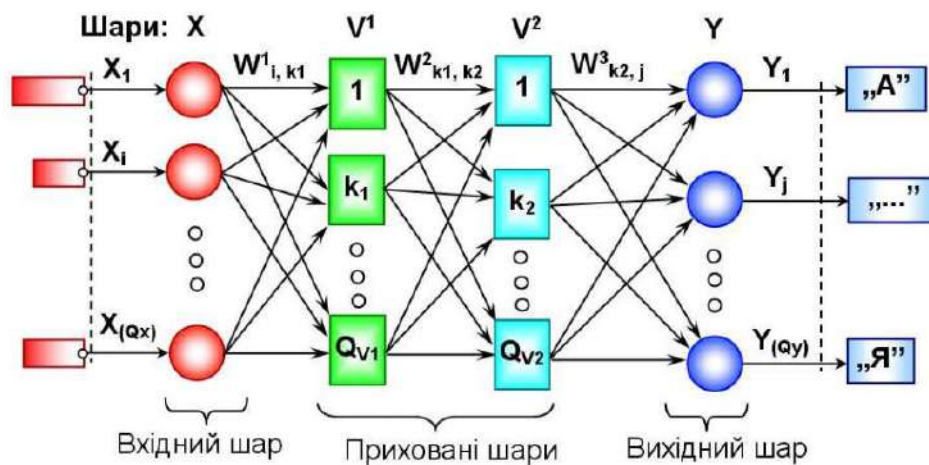


Рис 15. Схематичне зображення архітектури БШП

2. Програмна реалізація БШП за допомогою keras:

Спочатку завантажимо бібліотеку keras, набір даних, який складається із зображень одягу (MNIST Fashion Dataset) та поділимо його на тестові та навчальні шматки:

```
%tensorflow_version 2.x # This allows to select second version of TS # main libraries
import tensorflow as tf
from tensorflow import keras
```

```
import numpy as np
import matplotlib.pyplot as plt
```

```
fashion_mnist = keras.datasets.fashion_mnist # load dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data ()
```

У датасеті 60 тисяч зображень розміром 28 на 28, що дає 784 пікселя. Кожен із цих пікселів представляє колір у відтінках сірого 0-чорний і 255-білий.

Наступний етап – розмітимо датасет з 10 видів одягу:

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker',
               'Bag', 'Ankle boot']
```

Попередню обробку даних на цьому етапі буде спрощено до процедури нормалізації, яка змінює пікселі з діапазону [0,255] на [0,1]. Модель буде створена з використанням бібліотеки Keras. Модель послідовна, з трьома різними шарами.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)), # input layer (1)
    keras.layers.Dense(128, activation='relu'), # hidden layer (2)
    keras.layers.Dense(10, activation='softmax') # output layer (3)
])
```

Кількість нейронів у першому шарі співпадає з кількістю пікселів - 784. Другий шар прихований і має 128 нейронів, а отже 784x128 зв'язків з лінійною активаційною функцією. Останній шар має стільки ж нейронів, скільки класів

зображень – 10. Завершуючи модель, ми також повинні визначити тип оптимізації – adam та функція кросентропії. У моделі Keras тренування виконується функцією підгонки. Для навчання модельного навчального набору використовується його мітки (контрольоване навчання) і кількість епох. Чим більше епох навчена модель, тим точнішою вона може бути. Однак може виникнути і перетренованість – модель втратить можливість узагальнювати.

```
model.fit(train_images, train_labels, epochs=10)
```

Результат:

```
Epoch 1/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.4954 - accuracy: 0.8263
Epoch 2/10
1875/1875 [=====] - 8s 5ms/step - loss: 0.3780 - accuracy: 0.8652
Epoch 3/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.3400 - accuracy: 0.8760
Epoch 4/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.3171 - accuracy: 0.8845
Epoch 5/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.2990 - accuracy: 0.8901
Epoch 6/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.2844 - accuracy: 0.8939
```

З кожною епохою модель набуває точності, а це означає, що все більше зображень правильно класифікуються. Однак, щоб дійсно оцінити корисність моделі, нам потрібно було перевірити її на даних, які не використовувалися для навчання.

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=1) print("Test accuracy:", test_acc)
```

Результат:

```
313/313 [=====] - 1s 3ms/step - loss: 0.3304 - accuracy: 0.8841
Test accuracy: 0.8841000199317932
```

Точність на такому ж рівні, як у навчальній моделі, тому модель можна використовувати далі. Тож класифікуємо:

```
predictions = model.predict(test_images)
predictions[5]
```

Results:

```
array([1.8950767e-07, 9.9999976e-01, 7.4445838e-10,
0.0219599e-08, 2.6630920e-08, 5.2602244e-14, 1.5235547e-09,
7.1017318e-22,
8.1730155e-14, 7.6054418e-18], dtype=float32)
```

Результат відображається у вигляді 10 чисел від 0 до 1. Клас, який ми шукаємо, має найвищу цінність. Ми можемо знайти його за допомогою лінії:

```
np.argmax(predictions[5])
```

Результат: значення в нейроні високе 0,9999, тому класифікатор майже впевнений, що це штани, що можемо підтвердити, розглянувши зображення.

Завдання

1. Вивчити архітектуру багатошарового перцептрону;
2. Програмно реалізовувати задачу класифікації зображень згідно варіанту (додаток 2) за допомогою багатошарового перцептрону, змінюючи гіперпараметри архітектури, та провести дослідження 5 випадків;
3. Проаналізувати отримані результати.

Контрольні запитання

1. Які основні функції активації для багатошарового перцептрону?
2. Якими мають бути вхідний та вихідний шари?
3. Які методи навчання перцептрону є основними?

ЛІТЕРАТУРА: [5-10].

Лабораторна робота 10

КЛАСИФІКАЦІЯ НА ОСНОВІ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

Мета – набути практичних навичок реалізації архітектур згорткових нейронних мереж з метою класифікації даних.

Теоретичні відомості

1. Згорткові нейронні мережі:

Згорткові нейронні мережі (ЗНМ) в машинному навчанні — це клас глибинних штучних нейронних мереж прямого поширення, який успішно застосовувався до аналізу цифрових зображень. Вони складаються з шарів входу та виходу, а також із декількох прихованих шарів, якими є згорткові, агрегувальні, повнозв'язні шари та шари нормалізації.

Архітектура нейронної мережі. На рисунку 16 схематично зображено архітектуру нейронної мережі за шарами. Коротко охарактеризуємо призначення кожного шару.

Згорткові шари застосовують до входу операцію згортки, передаючи результат до наступного шару. Згортка імітує реакцію окремого нейрону на зоровий стимул. Кожен згортковий нейрон обробляє дані лише для свого рецептивного поля.

Хоч повнозв'язні нейронні мережі прямого поширення й можливо застосовувати як для навчання ознак, так і для класифікації даних, застосування цієї архітектури до зображень є непрактичним. Було би необхідним дуже велике число нейронів, навіть у поверхневій (протилежній до глибинної) архітектурі, через дуже великі розміри входу, пов'язані з зображеннями, де кожен піксель є відповідною змінною. Наприклад, повнозв'язний шар для (маленького) зображення розміром 100×100 має 10 000 ваг. Операція згортки дає змогу розв'язати цю проблему, оскільки вона зменшує кількість вільних параметрів, дозволяючи мережі бути глибшою за меншої кількості параметрів.

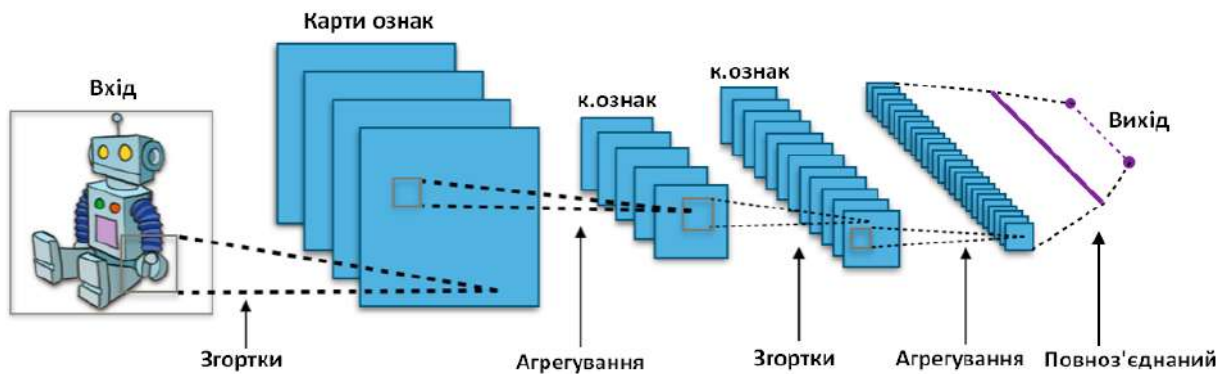


Рис 16. Схема архітектура ЗНМ

Наприклад, незалежно від розміру зображення, області замощування розміру 5×5 , кожна з одними й тими ж спільними вагами, вимагають лише 25 вільних параметрів. Таким чином, це розв'язує проблему зникання або вибуху градієнтів у тренуванні традиційних багатошарових нейронних мереж з багатьма шарами за допомогою зворотного поширення.

Агрегувальні шари. Згорткові мережі можуть включати шари локального або глобального агрегування, які об'єднують виходи кластерів нейронів одного шару до одного нейрону наступного шару. Наприклад, максимізаційне агрегування (англ. max pooling) використовує максимальне значення з кожного з кластерів нейронів попереднього шару. Іншим прикладом є усереднювальне агрегування (англ. average pooling), що використовує усереднене значення з кожного з кластерів нейронів попереднього шару.

Повнозв'язні шари з'єднують кожен нейрон одного шару з кожним нейроном наступного шару, що відповідає концепту багатошарового перцептрон (БШП).

Ваги. ЗНМ використовують спільні ваги в згорткових шарах, що означає, що для кожного рецептивного поля шару використовується один і той же фільтр (банк ваг); це зменшує обсяг необхідної пам'яті та поліпшує продуктивність.

2. *Програмна реалізація згорткової нейронної мережі з метою класифікації:*

В якості тренувального набору даних використаємо Fashion-MNIST датасет, як і в попередній лабораторній роботі. Розіб'ємо датасет у співвідношенні 70% (42000 екземплярів) / 30% (18000 екземплярів) використовуючи 6000 ескізів, 10000 залишимо на набір валідації. Виконаємо препроцесинг даних.

```
train_X, test_X, train_Y, test_Y = train_test_split(X, Y, test_size=0.3)
```

```
print('Тренувальний набір: ', train_X.shape, train_Y.shape)
```

```
print('Тестовий набір: ', test_X.shape, test_Y.shape)
```

```
Тренувальний набір: (42000, 28, 28) (42000,)
```

```
Тестовий набір: (18000, 28, 28) (18000,)
```

```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255.
test_X = test_X / 255.
```

```
train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)
train_X.shape, test_X.shape
```

```
((42000, 28, 28, 1), (18000, 28, 28, 1))
```

```
import keras
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
```

```
batch_size = 64
epochs = 20
num_classes = 10
```

Використаємо наступні кроки для побудови архітектури

- три шари згортки: перший 32 3*3 фільтри, другий 64 3*3 фільтри, третій 128 3*3 фільтри
- Після кожного шару згортки додамо шар пулінгу 2*2.

У результаті отримаємо:

1. Input 28*28*1
2. Convolution 32-3*3 filter
3. Max pooling 2*2
4. Convolution 64-3*3 filter
5. Max pooling 2*2
6. Convolution 128-3*3 filter
7. Max pooling 2*2
8. Flatten
9. Dense layer 128 units
10. Output layer 10 units

Імпортуємо необхідні бібліотеки для побудови нейронної мережі.

Встановимо параметри навчання.

```
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

print('Оригінальний клас:', train_Y[1])
print('Після преппроцесінгу:', train_Y_one_hot[1])
```

```
Оригінальний клас: 3
Після преппроцесінгу: [0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

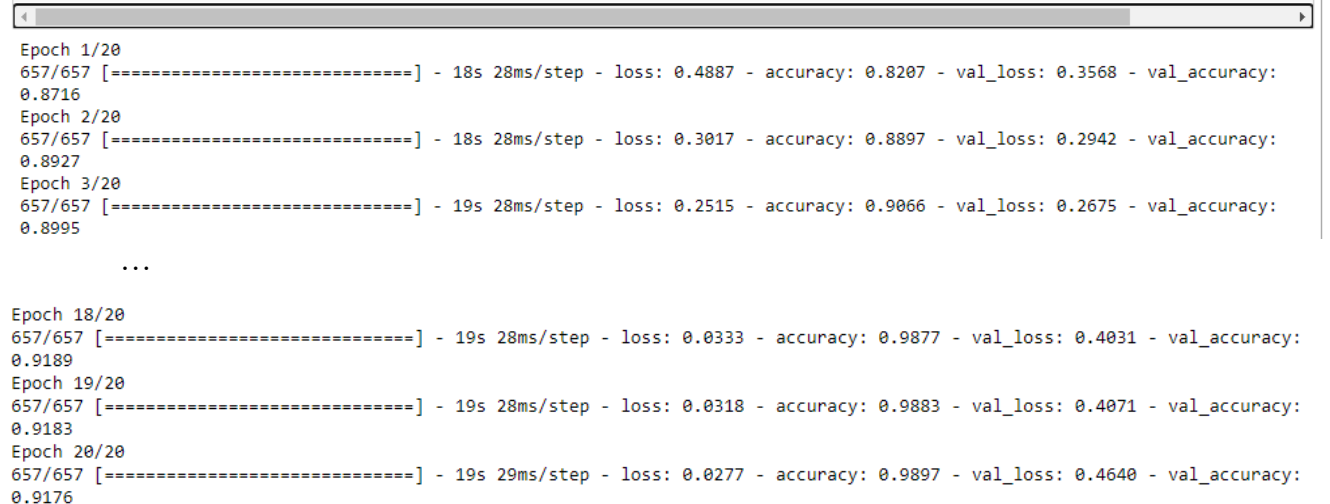
Опишемо нейронну мережу згідно алгоритму. Скомпілюємо нейронну мережу. Задамо функцію втрат – крос-ентропія, вона використовується для мульти-класової класифікації. Задаймо метрику, за якою хочемо спостерігати під час навчання мережі – точність.

```
fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3, 3), activation='linear', input_shape=(28,28,1), padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2), padding='same'))
fashion_model.add(Conv2D(64, (3, 3), activation='linear', padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
fashion_model.add(Conv2D(128, (3, 3), activation='linear', padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(Dense(num_classes, activation='softmax'))
```

```
fashion_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(), metrics=['accuracy'])
```

На тренуємо мережу на тренувальних даних, використовуючи 20 епох.

```
: fashion_train = fashion_model.fit(train_X, train_Y_one_hot, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(test_X, test_Y_one_hot))
```



```
Epoch 1/20
657/657 [=====] - 18s 28ms/step - loss: 0.4887 - accuracy: 0.8207 - val_loss: 0.3568 - val_accuracy: 0.8716
Epoch 2/20
657/657 [=====] - 18s 28ms/step - loss: 0.3017 - accuracy: 0.8897 - val_loss: 0.2942 - val_accuracy: 0.8927
Epoch 3/20
657/657 [=====] - 19s 28ms/step - loss: 0.2515 - accuracy: 0.9066 - val_loss: 0.2675 - val_accuracy: 0.8995
...
Epoch 18/20
657/657 [=====] - 19s 28ms/step - loss: 0.0333 - accuracy: 0.9877 - val_loss: 0.4031 - val_accuracy: 0.9189
Epoch 19/20
657/657 [=====] - 19s 28ms/step - loss: 0.0318 - accuracy: 0.9883 - val_loss: 0.4071 - val_accuracy: 0.9183
Epoch 20/20
657/657 [=====] - 19s 29ms/step - loss: 0.0277 - accuracy: 0.9897 - val_loss: 0.4640 - val_accuracy: 0.9176
```

Як бачимо, точність тренувальних даних 99% і втрати дуже низькі. Проте втрати набору валідації (тестового) – 0.4640 високі і точність 91.76%. Це вказує на перенавчання моделі.

Використаємо «відкладені» данні, які мережа ще не бачила, для оцінки точності.

```
valid_X = X1.reshape(-1, 28,28, 1)
valid_X = valid_X.astype('float32')
valid_X = valid_X / 255.
valid_Y_one_hot = to_categorical(Y1)
```

```
test_eval = fashion_model.evaluate(valid_X, valid_Y_one_hot, verbose=0)
```

```
print('Validation loss:', test_eval[0])
print('Validation accuracy:', test_eval[1])
```

Validation loss: 0.5302691459655762

Validation accuracy: 0.9142000079154968

Візуалізуємо зміну точності і втрат тестового і тренувального набору.

```
accuracy = fashion_train.history['accuracy']
val_accuracy = fashion_train.history['val_accuracy']
loss = fashion_train.history['loss']
val_loss = fashion_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Testing accuracy')
plt.title('Training and testing accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Testing loss')
plt.title('Training and testing loss')
plt.legend()
plt.show()
```

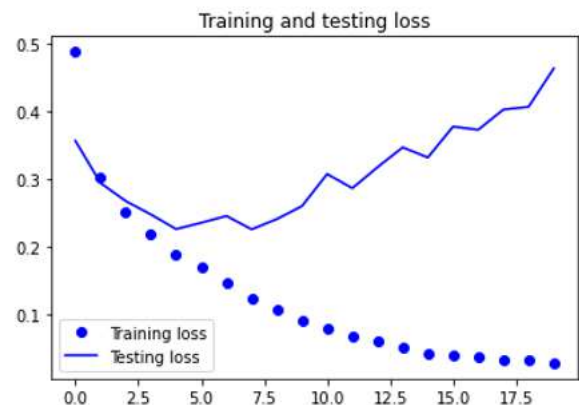
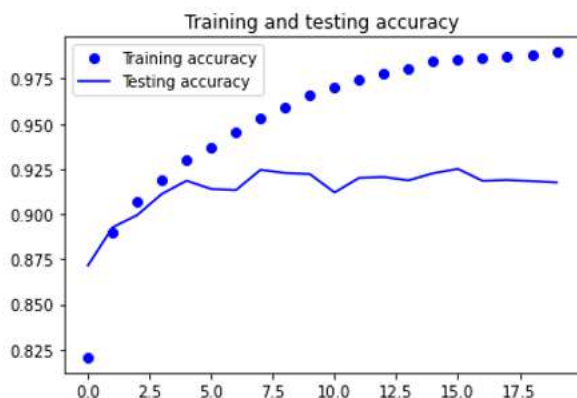


Рис 17. Графіки зміни точності (ліворуч) та втрат (праворуч) залежно від епохи навчання

З наведених на Рис.17 двох графіків видно, що точність перевірки зазнала стагнації через 4-5 епохи і рідко зростала в певні епохи. На початку точність тестового набору лінійно зростала із втратами, але потім вона не надто зростала.

Втрата при тестуванні показує, що присутнє перенавчання, подібно точності перевірки вона лінійно зменшувалась, але через 4-5 епохи почала зростати. Це означає, що модель намагалася запам'ятати дані і досягла успіху.

Спробуємо змінити нашу модель. Додамо шар, що має подолати проблему перенавчання - dropout. Він випадковим чином вилучає частку нейронів під час тренувального процесу, зменшуючи залежність від тренувань. Це не дозволяє мережі запам'ятовувати навчальні данні.

Оновимо схему архітектури мережі.

```
: fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3, 3), activation='linear', padding='same', input_shape=(28, 28, 1)))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2), padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(64, (3, 3), activation='linear', padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(128, (3, 3), activation='linear', padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
fashion_model.add(Dropout(0.4))
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(Dropout(0.3))
fashion_model.add(Dense(num_classes, activation='softmax'))
```

Скомпілюємо модель та запустимо процес навчання (параметри незмінні).

```
...
Epoch 18/20
657/657 [=====] - 22s 33ms/step - loss: 0.2070 - accuracy: 0.9210 - val_loss: 0.2078 - val_accuracy:
0.9254
Epoch 19/20
657/657 [=====] - 21s 33ms/step - loss: 0.2043 - accuracy: 0.9239 - val_loss: 0.2069 - val_accuracy:
0.9248
Epoch 20/20
657/657 [=====] - 22s 33ms/step - loss: 0.2007 - accuracy: 0.9253 - val_loss: 0.2064 - val_accuracy:
0.9254
Epoch 21/20
557/657 [=====>.....] - ETA: 2s - loss: 0.4170 - accuracy: 0.8466
```

Схоже, що додавання шару Dropout зменшило кількість втрат на наборі валідації. У даному випадку, втрати та точність тестового набору синхронізуються з втратами і точністю тренувального. Модель не є надмірно навченою: втрати перевірки зменшуються і не збільшуються, і між точністю навчального та тестового наборів немає великого розриву.

Завдання

1. Вивчити архітектуру згорткової нейронної мережі;
2. Програмно реалізувати задачу класифікації зображень згідно варіанту (додаток 2) за допомогою згорткової нейронної мережі, змінюючи гіперпараметри архітектур, та провести дослідження 5 випадків;
3. Проаналізувати отримані результати.

Контрольні запитання

1. Проведіть огляд типової архітектури згорткової нейронної мережі?
2. Які прийоми запобігання перенавчання згорткової нейронної мережі існують?
3. Які прийоми регуляції згорткової нейронної мережі існують?

4. Що таке згортковий шар та шар підвибірки у згортковій нейронній мережі?

5. Що таке ReLU у згортковій нейронній мережі?

ЛІТЕРАТУРА: [5-10].

Додаток 1. Варіанти завдань для виконання лабораторних робіт 1-8.

№ варіанту	Назва датасету	Посилання
1	Індикатори світової економіки	https://www.kaggle.com/datasets/prasad22/global-economy-indicators
2	Дані фондової біржі	https://www.kaggle.com/mattiuzc/stock-exchange-data
3	Щоденні дані про погоду за календарний рік	https://www.kaggle.com/vishalvjoseph/weather-dataset-for-covid19-predictions
4	Глобальне заміщення енергії з 1983 по 2022 рік	https://www.kaggle.com/datasets/mohamedyosef101/global-energy-substitution-from-1983-to-2022
5	Продажі нерухомості 2001-2020 штат Коннектикут	https://www.kaggle.com/datasets/anoopjohny/real-estate-sales-2001-2020-state-of-connecticut
6	Якість сну	https://www.kaggle.com/datasets/risakashiwabara/japanquality-of-sleep
7	Олімпійський набір даних	https://www.kaggle.com/datasets/mohammadkaiftahir/olympics-dataset
8	Набір даних статистики пожеж	https://www.kaggle.com/datasets/sujaykapadnis/fire-statistics-dataset
9	Медичне страхування та зловживання	https://www.kaggle.com/datasets/willianoliveiragibin/mendical-insurance-and-malpractice
10	Пенсійне забезпечення в США (1979-2019)	https://www.kaggle.com/datasets/asaniczka/pension-coverage-in-the-usa-1973-2022

Додаток 2. Варіанти завдань для виконання лабораторних робіт 9-10

№ варіанту	Назва датасету	Посилання
1	tomatOD	https://www.kaggle.com/datasets/nexuswho/tomatod
2	Коти і собаки	https://www.kaggle.com/datasets/bhavikjikadara/dog-and-cat-classification-dataset
3	Рентген колінного суглоба	https://www.kaggle.com/datasets/trainingdatapro/knee-x-rays-4-stages-of-osteoarthritis
4	Рентген зображення грудної клітки	https://www.kaggle.com/datasets/trainingdatapro/chest-xray-17-diseases
5	Сегментація судин	https://www.idiap.ch/software/bob/docs/bob/bob.db.drive/stable/index.html
6	War_tech_v2.0 detection objects	https://www.kaggle.com/datasets/gon213/war-tech-v2-0-by-gontech
7	Набір даних Malhari	https://www.kaggle.com/datasets/jocelyndumlao/malhari-dataset
8	Грудна пневмонія	https://www.kaggle.com/datasets/thomasdubail/chest-pneumonia-256x256
9	Набір даних зображень листя	https://www.kaggle.com/datasets/oliverdaubney/unstructured-leaf-images-dataset
10	Набір даних Stanford Cars	https://www.kaggle.com/datasets/jessicali9530/stanford-cars-dataset

СПИСОК ЛІТЕРАТУРИ

1. Tutorial NumPy. Режим доступу: <https://numpy.org/doc/stable/index.html>.
2. Tutorial Pandas. Режим доступу: <https://pandas.pydata.org/docs/index.html>.
3. Tutorial Scikit-learn. Режим доступу: <https://scikit-learn.org/stable/index.html>
4. Natural Language Toolkit. Режим доступу: <https://www.nltk.org/>
5. Tutorial TensorFlow. Режим доступу: <https://www.tensorflow.org/>
6. Burkov A. The hundred-page machine learning book / Andrey Burkov Режим доступу: <https://leanpub.com/theMLbook>.
7. Shalev-Shwartz S., Ben-David S. Understanding machine learning. / Shai Shalev-Shwartz , Shai Ben-David. — Cambridge University Press: New York, 2014. — 416 pp.
8. Geron A. Hands-on machine learning with Scikit-learn, Keras, and TensorFlow / Aurelien Geron. — O'Reilly Media, 2019. — 510 p.
9. Laros D.T., Laros C.D. Discovering knowledge in data/ Daniel T. Laros and Chantal D. Laros – Wiley&Sons: New Jersey, 2014. — 332 p.
10. Brownlee J. Machine learning mastery with Python /J. Brownlee. — Machine Learning Mastery, 2016. — 179 p.

Навчально – методичне видання

ПРОГРАМНІ ТЕХНОЛОГІЇ МАШИННОГО НАВЧАННЯ

Лабораторний практикум
для студентів спеціальності
121 «Інженерія програмного забезпечення»