Практичний кейс 1

Вступ до машинного навчання**

Тема: Огляд основних концепцій машинного навчання, приклади застосувань.

Мета: Ознайомитися з основними поняттями та етапами процесу машинного навчання, інтегрованим середовищем Jupyter Notebook та використанням Python бібліотек для ML.

Завдання:

1. Налаштувати хмарний сервіс Google Colab для виконання Jupyter Notebook з підтримкою GPU та можливістю ділитися з іншими.
2. Ознайомитися з документацією Jupyter Notebook.
3. Виконати огляд наборів даних (Iris, MNIST, Titanic).
4. Провести базову візуалізацію даних за допомогою Matplotlib та Seaborn.
5. Виконати ручне розділення даних на тренувальну і тестову вибірки.

Хід роботи:

Завдання 1: Налаштування Google Colab для виконання Jupyter Notebook з підтримкою GPU та можливістю ділитися з іншими

Кроки виконання завдання

1.1. Створення нового Jupyter Notebook

a. Перейдіть на сайт Google Colab.
b. На вітальній сторінці оберіть **“New notebook”** у верхньому правому куті.

Тепер ви можете писати код у блоках (комірках), як у звичайному Jupyter Notebook, використовуючи **Python** як основну мову програмування.
Для виконання коду в комірці натисніть на неї і використайте комбінацію клавіш **Shift + Enter** або натисніть на кнопку “Run” біля комірки.

1.2. Імпорт даних

a. Завантаження файлів з використанням бібліотеки `google.colab`**
Ви можете завантажити файли з вашого комп’ютера, використовуючи команду `files.upload()` з бібліотеки `google.colab`:

```
# Імпортуємо модуль files з бібліотеки google.colab
from google.colab import files

# Функцію upload() відкриє діалог завантаження файлів
uploaded = files.upload()

# Завантаженні файли будуть доступні як ключі словника uploaded
# Щоб переглянути імена завантажених файлів, виконайте:
for filename in uploaded.keys():
    print('Завантажено файл "{name}" з розміром {length} байт'.format(
        name=filename, length=len(uploaded[filename])))
```

Після завантаження файли будуть доступні у файловій системі Colab, і ви зможете прочитати їх за допомогою функцій бібліотеки `pandas`. Для файлів CSV, Excel та SQL потрібно використовувати свої функції:

```
import pandas as pd
import sqlite3 # Драйвер для роботи з потрібною базою даних

# Читаємо CSV файл
df_csv = pd.read_csv('data/data.csv')
print("CSV Data:")
print(df_csv.head())

# Читаємо Excel файлу
# За замовчуванням читається перший аркуш
f_excel = pd.read\_excel('data.xlsx')
print(df\_excel.head())
# Читаємо дані з аркуша "Sheet2"
df_excel_sheet2 = pd.read\_excel('data.xlsx', sheet\_name='Sheet2')
        print(df_excel_sheet2.head())

# Читаємо дані з SQLite бази даних
# З'єднання з базою даних.
conn = sqlite3.connect('data/database.db')  #
query = "SELECT * FROM users"
#  Виконання SQL-запиту
df_sql = pd.read_sql_query(query, conn)
conn.close()

print("\nSQL Data:")
print(df_sql.head())
```

1.3. Підключення Google Drive

Файли, завантажені через `files.upload()`, будуть доступні лише під час поточної сесії. Якщо сесія завершиться, вам доведеться завантажити їх знову. Якщо ви плануєте часто використовувати один і той же набір даних, зручніше зберігати їх на Google Drive та монтувати його в Colab.
Якщо потрібно працювати з файлами на Google Drive, ви можете підключити свій диск до Colab за допомогою наступної команди:

```
from google.colab import drive
drive.mount('/content/drive')
```

Після успішного монтування ви зможете отримати доступ до файлів на Google Drive, наприклад:

```
import pandas as pd

# Шлях до файлу у Google Drive
file_path = '/content/drive/MyDrive/Stocks.csv'
df = pd.read_csv(file_path)
print(df.head())
```

1.4. Використання GPU або TPU

Ви можете використовувати графічний процесор (GPU) або навіть тензорний процесор (TPU) для прискорення обчислень. Для цього:

a. Перейдіть у меню **“Edit” > “Notebook settings”** (або **“Runtime” > “Change runtime type”**).
b. Оберіть **“GPU”** у полі **“Hardware accelerator”**.

c. Натисніть **“Save”**.

1.5. Налаштування спільного доступу

a. Натисніть кнопку **“Share”** у верхньому правому куті ноутбука.
b. Налаштуйте права доступу (наприклад, “Anyone with the link can view/comment/edit” залежно від потреб) та скопіюйте посилання для спільного доступу.

1.6. Збереження та експорт

a. **Збереження на Google Drive**
Google Colab автоматично зберігає ваші ноутбуки на вашому Google Drive. Щоб зберегти копію ноутбука на Google Drive, виберіть **File** > **Save a copy in Drive**.

b. **Збереження на Github**
У меню **File** Google Colab доступні дві команди, які дозволяють зберігати копії вашого блокноту безпосередньо на GitHub:

- **Save a copy as a GitHub Gist**
GitHub Gist – це сервіс від GitHub для зберігання та публікації окремих фрагментів коду або невеликих файлів. Gist часто використовується для швидкого обміну кодом, демонстрацій чи зразків.
Використовуючи цю команду, ви створюєте новий gist, до якого автоматично завантажується копія вашого Colab блокноту. Він зберігається як окремий документ у сервісі GitHub Gist.

- **Save a copy in GitHub**
Ця команда дозволяє зберегти копію вашого блокноту безпосередньо в репозиторії GitHub. Ви можете вибрати конкретний репозиторій, де буде збережений ваш Colab блокнот, а також вказати гілку (branch) та шлях до файлу. Такий підхід більше підходить для ведення історії версій та інтеграції з іншими файлами проекту, адже блокнот стане частиною більшого репозиторію на GitHub.

  Сценарій використання:

  Ви розробляєте проект, у якому використовуєте Google Colab для експериментів та аналізу так, щоб ваш блокнот зберігався разом з іншим кодом проекту. Для цього:

  - Оберіть **File > Save a copy in GitHub**.
  - Вкажіть репозиторій, в якому потрібно зберегти блокнот.
  - Налаштуйте commit message.
  - Після підтвердження Colab завантажить ваш блокнот у вказаний репозиторій, і він стане частиною вашого проекту на GitHub.

c. **Експорт**
Ви можете завантажити ваш ноутбук у форматі `.ipynb` або експортувати його як `.py` або `.html`. Для цього виберіть **File** > **Download** і виберіть формат.

1.7. Управління версіями документів

a. **Вбудована історія версій у Google Colab**

За замовчуванням Google Colab автоматично зберігає ноутбуки на вашому Google Drive. Це дозволяє легко повернутися до попередніх версій, оскільки кожна зміна записується. Ви можете переглянути історію змін, обравши в меню **«File» > «Revision history»**.
Якщо ваша робота носить експериментальний характер і вам не потрібен детальний контроль версій, можна обійтися вбудованою історією змін у Google Colab, збереженою на Google Drive.

b. **Інтеграція з GitHub**

Для серйозних проектів, командної роботи або публікації коду рекомендується створити репозиторій на GitHub (або іншій системі контролю версій) для гнучкого управління гілками, злиттями та іншими аспектами Git. Далі потрібно клонувати цей репозиторій у Google Colab і налаштувати Git (ім'я користувача, email). Після цього можна редагувати ноутбуки, перевіряти статус, додавати коміти, пушити зміни і створювати пулреквести для інтеграції змін за допомогою команд Git. Щоб працювати з Git в Google Colab, потрібно скористатись командною оболонкою (терміналом) через клітинки з виконанням команд оболонки (починаються зі знаку !), наприклад:

```
!git clone https://YOUR_GITHUB_TOKEN@github.com/yourusername/my-
        notebook.git
```

Завдання 2: Ознайомлення з документацією Jupyter Notebook

Кроки виконання завдання

2.1. Ознайомлення з документацією

a. На сторінці документації Jupyter Notebook ознайомтеся з розділами:

- The Jupyter Notebook
- User interface components

b. Ознайомтеся з налаштуваннями та конфігурацією Jupyter Notebook.
c. Створити загальний список **магічних команд** в Jupyter Notebook, вказавши призначення для кожної.

2.2. Відповіді на контрольні питання

Надайте відповіді на контрольні питання:

```
- Що таке Jupyter Notebook і які його основні призначення?
- Які галузі застосування знаходить Jupyter Notebook (наприклад,
аналіз даних, машинне навчання, наукові дослідження)?
- Які мови програмування підтримуються в Jupyter Notebook?
- Як запустити Jupyter Notebook з командного рядка?
- Які вимоги до системи та залежностей потрібно враховувати при
встановленні?
- Які основні елементи інтерфейсу Jupyter Notebook?
- Які типи комірок доступні в Jupyter Notebook і для чого кожна з них
використовується (код, Markdown, Raw)?
- Як додати, видалити, перемістити або редагувати комірки?
- Як створювати документ із текстовими блоками, використовуючи
Markdown?
- Яким чином можна вставляти формули LaTeX у Markdown-комірки?
- Які бібліотеки для візуалізації даних можна використовувати в
Jupyter Notebook?
- Що таке інтерактивні віджети (ipywidgets) і як вони інтегруються в
Jupyter Notebook?
- Які приклади використання інтерактивних елементів у аналізі даних
можна навести?
- Яким чином встановлюються та налаштовуються розширення для Jupyter
Notebook?
- Які формати експорту підтримуються?
- Як за допомогою nbconvert конвертувати Notebook в інші формати?
- Які налаштування збереження можна застосувати для підтримки
версіонування документа?
```

- Що таке магічні команди і як вони спрощують роботу з ноутбуками?
- Наведіть приклади магічних команд і як їх використовувати?
- Як отримати список всіх доступних магічних команд у середовищі?
- Які можливості налагодження коду надає Jupyter Notebook?
- Як використовувати інтегровані інструменти для моніторингу продуктивності (наприклад, вимірювання часу виконання коду)?
- Які оптимізувати роботу з великими даними в Jupyter Notebook?

Завдання 3: Огляд наборів даних (Iris, MNIST, Titanic)

Кроки виконання завдання

3.1. Iris

3.1.1. Загальний опис.

Iris – набір даних про види ірисів з 150 зразками, що містить 4 числових ознаки.

- **Джерело:** Вбудований у `sklearn.datasets`.
- **Кількість записів:** 150 спостережень.
- **Ознаки**:

1. **Довжина чашолистка (sepal length):** Вимірюється в сантиметрах.
2. **Ширина чашолистка (sepal width):** Вимірюється в сантиметрах.
3. **Довжина пелюстки (petal length):** Вимірюється в сантиметрах.
4. **Ширина пелюстки (petal width):** Вимірюється в сантиметрах.

- **Класи (мітки)**:

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

3.1.2. Завантаження та ознайомлення з даними

a. Імпортуйте необхідні бібліотеки:

- `pandas` для роботи з даними.
- `numpy` для математичних операцій.
- `matplotlib` та `seaborn` для побудови графіків.
- `sklearn.datasets` для завантаження набору даних Iris.

b. Завантажте набір даних:

- Використайте функцію `load_iris()` з бібліотеки `sklearn.datasets`.
- Створіть DataFrame за допомогою `pandas` для кращого аналізу.

Приклад:

```
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)
df['target'] = iris.target df['species'] = df['target'].apply(lambda
        x: iris.target\_names\[x])
print(df.head())
```

c. Перегляньте перші кілька рядків даних, використовуючи метод `head()`. Перегляньте розмірність даних, типи даних та опис ознак.

3.1.3. Проведення попереднього аналізу даних (Exploratory Data Analysis)

a. Перевірте кількість зразків і ознак для кожного зразка і їх форму у наборі даних:

```
data = iris.data
target = iris.target
print(data.shape)
print(target.shape)
```

Поясніть результат виводу цього коду, враховуючи, що `iris.data` є двовимірним масивом, де кожен рядок представляє окремий зразок, а кожен стовпець – характеристику цього зразка, а `iris.target` є одновимірним масивом, де кожен елемент є міткою (наприклад, номером виду рослини), який відповідає зразку з відповідного рядка в `iris.data`.

b. Використайте метод `describe()` для отримання основних статистик (середнє, стандартне відхилення, мінімальне та максимальне значення).

c. Перевірте наявність пропущених значень за допомогою методу `isnull().sum()`. Переконайтесь, що дані не містять пропусків.

3.2. MNIST

3.2.1. Загальний опис.

MNIST (Modified National Institute of Standards and Technology) – це класичний набір даних, що складається з зображень рукописних цифр. Кожне зображення представляється як матриця чисел від 0 до 255 (значення пікселів), де 0 означає чорний колір, а 255 — білий.

- **Джерело:** Можна завантажити через `tensorflow.keras.datasets`.
- **Кількість записів:** 60,000 тренувальних із відповідними мітками (цифрами від 0 до 9) і 10,000 тестових зразків із відповідними мітками для оцінки якості моделі.

3.2.2. Завантаження та ознайомлення з даними

a. **Імпортуйте необхідні бібліотеки:**

- `matplotlib` для побудови зображень і побудови графіків
- `tensorflow.keras.datasets` для завантаження набору даних MNIST

b. **Завантажте набір даних:**

Приклад:

```
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Завантаження набору даних MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Вибір першого зображення та відповідної мітки
image = x_train[0]
label = y_train[0]

# Вивід мітки у консоль
print("Мітка зображення:", label)

# Візуалізація зображення
plt.imshow(image, cmap='gray')
plt.title(f"Мітка: {label}")
```

```
plt.axis('off')  # Прибираємо осі для кращої візуалізації
plt.show()
```

3.2.3. Проведення попереднього аналізу даних (Exploratory Data Analysis)

a. Отримайте форми тренувальних і тестових зображень і міток з допомогою методу `shape` (наприклад, `x_train.shape`).

У контексті машинного навчання та обробки зображень поняття **“форма”** (shape) використовується для опису розмірності масивів даних, які представляють як зображення, так і відповідні мітки.
Форма тренувальних зображень – це характеристика, яка вказує, якою є структура (розмірність) даних, що представляють зображення, у вигляді багатовимірного масиву. Форма описує, скільки зображень є в наборі, а також розміри кожного зображення і кількість каналів кольорів.
Наприклад, якщо в наборі 60 000 зображень, де кожне зображення має розмір 28x28 пікселів і є чорно-білим (1 канал), то форма може бути представлена як `(60000, 28, 28, 1)`

Форма тренувальних міток (labels) вказує, як представлені дані, що відповідають класовим міткам або значенням цільової змінної для кожного зображення.
Наприклад, якщо кожне з 60 000 зображень має відповідну мітку у вигляді цілого числа, то форма міток може бути такою `(60000,)`

b. Нормалізуйте пікселі тренувальних і тестових зображень (приведення значень пікселів до діапазону [0, 1]) з допомогою методу `astype` (наприклад, `x_train.astype('float32')`).

Переконайтеся, що пікселі зображень нормалізовано:

```
image = x\_train\_normalized[0]
print(image)
```

3.3. Titanic

3.3.1. Загальний опис

Набір даних Titanic – це класичний набір даних, який містить інформацію про пасажирів лайнера «Titanic».

- **Джерело:** Доступний на Kaggle або через бібліотеку `seaborn`.
- **Основні ознаки (колонки):**
  - **PassengerId**
    Опис: Унікальний ідентифікатор кожного пасажира.
    Тип: Ціле число.
  - **Survived**
    Опис: Цільова змінна, що позначає, чи вижив пасажир (0 – не вижив, 1 – вижив).
    Тип: Бінарна категорія.
  - **Pclass**
    Опис: Клас пасажира (1, 2 або 3), що відображає соціально-економічний статус.
    Тип: Категорійне числове значення.
  - **Name**
    Опис: Ім’я та прізвище пасажира, інколи містить титули (наприклад, Mr., Mrs., Miss), що можуть бути корисними для додаткового аналізу.
    Тип: Текст.

- **Sex**
  Опис: Стать пасажира (male/female).
  Тип: Категорія.

- **Age**
  Опис: Вік пасажира в роках.
  Тип: Дійсне число.
  Особливості: Може містити пропущені значення.

- **SibSp**
  Опис: Кількість братів/сестер або подружжя пасажира на борту.
  Тип: Ціле число.

- **Parch**
  Опис: Кількість батьків чи дітей пасажира на борту.
  Тип: Ціле число.

- **Ticket**
  Опис: Номер квитка. Може містити як числову, так і текстову інформацію (іноді з кодами).
  Тип: Текст.

- **Fare**
  Опис: Сума, сплачена за проїзд.
  Тип: Дійсне число.

- **Cabin**
  Опис: Номер каюти. Дані можуть бути неповними, а також містять літерні позначення, що вказують на розташування.
  Тип: Текст.

- **Embarked**
  Опис: Порт посадки (C – Cherbourg, Q – Queenstown, S – Southampton).
  Тип: Категорія.

- **Особливості набору даних:**

  - **Пропущені значення:**
    Деякі ознаки (наприклад, *Age* та *Cabin*) можуть містити пропущені значення, що потребують попередньої обробки.

  - **Категоріальні та числові дані:**
    Набір містить як числові (наприклад, *Fare*, *Age*), так і категоріальні дані (наприклад, *Sex*, *Embarked*).

3.3.2. Завантаження та ознайомлення з даними

a. Завантажте набір даних за допомогою бібліотеки seaborn або pandas.

Приклад:

```
import seaborn as sns
titanic = sns.load_dataset('titanic')
print(titanic.head())
```

або (якщо перший варіант не працює)

```
import pandas as pd
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.c
titanic = pd.read_csv(url)
print(titanic.head())
```

b. Перевірте розміри набору даних та інформацію про кожну колонку (типи даних, кількість пропущених значень).

Приклад:

```
print(data.shape)
print(data.info())
print(data.describe())
```

3.3.3. Попередня обробка даних

a. Обробіть пропущені значення:

- Заповніть пропущені значення у *Age* (наприклад, середнім або медіанним значенням).
- Розгляньте можливість видалення або кодування пропущених значень для *Cabin*.

```
# Заповнення пропущених значень для Age медіаною
data['Age'].fillna(data['Age'].median(), inplace=True)
```

b. Перетворіть категоріальні ознаки (наприклад, *Sex*, *Embarked*) у числові значення, використовуючи one-hot encoding або label encoding.

```
data = pd.get_dummies(data, columns=['Sex', 'Embarked'],
        drop_first=True)
```

(c) Перевірте можливість створення нових ознак:

- Виділення титулу з поля *Name*.
- Обчислення розміру родини на борту (сума *SibSp* та *Parch*).

```
# Приклад створення нової ознаки "FamilySize"
data['FamilySize'] = data['SibSp'] + data['Parch'] + 1
```

Завдання 4: Візуалізація даних за допомогою Matplotlib та Seaborn

Кроки виконання завдання

Побудуйте кілька базових графіків для отримання уявлення про структуру даних з використанням бібліотек Matplotlib та Seaborn.

4.1. Для набору Iris

Завантаження набору:

```
import matplotlib.pyplot as plt
import seaborn as sns

iris = sns.load\_dataset("iris")
```

a. Побудуйте гістограми розподілу для кожної числової ознаки (sepal length, sepal width, petal length, petal width) з набору Iris.

Приклад:

```
sns.load_dataset("iris")
plt.hist(iris.data[:, 0], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Сепальна довжина (cm)')
plt.ylabel('Кількість зразків')
plt.title('Розподіл сепальної довжини для набору Iris')
plt.show()
```

b. Побудуйте накладені гістограми для порівняння розподілів однієї ознаки для різних видів ірисів. Накладені гістограми дозволяють порівняти, як змінюється розподіл значень однієї ознаки для кожного виду ірису.

Приклад:

```
plt.figure(figsize=(8, 6))
species = iris['species'].unique()
colors = ['red', 'green', 'blue']

for sp, col in zip(species, colors):
    subset = iris[iris['species'] == sp]
    plt.hist(subset['sepal_length'], bins=15, alpha=0.5, label=sp,
        color=col, edgecolor='black')

plt.title('Гістограма довжини чашолистка за видами')
plt.xlabel('sepal_length')
plt.ylabel('Частота')
plt.legend()
plt.show()
```

c. Побудуйте розділені гістограми для детального аналізу розподілів по групах. Для кожного виду створється окрема гістограма, що дозволяє детально порівняти розподіли без накладення.

Приклад:

```
g = sns.FacetGrid(iris, col="species", height=4, aspect=1)
g.map(plt.hist, "sepal_length", bins=15, color='coral',
        edgecolor='black')
plt.show()
```

d. Побудуйте гістограм із накладеною кривою оцінки щільності розподілу (kernel density estimation, KDE). Це дозволяє отримати більш гладку оцінку розподілу.

Приклад:

```
plt.figure(figsize=(8, 6))
sns.histplot(iris['petal_length'], bins=15, kde=True, color='purple',
        edgecolor='black')
plt.title('Гістограма та KDE для довжини пелюстки')
plt.xlabel('petal_length')
plt.ylabel('Частота')
plt.show()
```

e. Побудуйте кумулятивну гістограму для аналізу накопичуваних частот. Кумулятивні гістограми показують накопичуваний розподіл даних, що може бути корисно для оцінки процентилів.

Приклад:

```
plt.figure(figsize=(8, 6))
plt.hist(iris['sepal_width'], bins=15, cumulative=True, color='teal',
        edgecolor='black')
plt.title('Кумулятивна гістограма для sepal_width')
plt.xlabel('sepal_width')
plt.ylabel('Накопичена частота')
plt.show()
```

f. Створіть парну діаграму (pairplot) для перегляду взаємозв’язків між ознаками з розподілом за класами.

Приклад:

```
# Побудова pairplot
sns.pairplot(df, hue='species', markers=["o", "s", "D"])
plt.show()
```

4.2. Для набору Titanic

a. Побудуйте парний графік (pairplot) для набору Titanic

```
import seaborn as sns
titanic = sns.load_dataset('titanic')
sns.pairplot(titanic.dropna(), hue='survived', vars=['age', 'fare'])
plt.suptitle('Парний графік ознак для Titanic', y=1.02)
plt.show()
```

b. Створіть діаграму розподілу кількості пасажирів, що вижили та не вижили.

Приклад:

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.countplot(x='survived', data=data)
plt.title('Розподіл виживання')
plt.show()
```

(c) Побудуйте графіки залежності виживання від ключових ознак:

- Стать (*Sex*) – виживання за статтю
- Клас пасажира (*Pclass*) – виживання за класом

Приклад:

```
# Виживання за статтю
sns.countplot(x='Sex', hue='Survived', data=data)
plt.title('Виживання за статтю')
plt.show()
```

d. Побудуйте гістограми для числових змінних *Age* і *Fare*, щоб дослідити їх розподіл.

```
data['Age'].hist(bins=30)
plt.title('Розподіл віку')
plt.xlabel('Вік')
plt.ylabel('Кількість пасажирів')
plt.show()
```

Завдання 5: Ручне розділення даних на тренувальну і тестову вибірки

Кроки виконання завдання

Необхідно вручну розділити дані на тренувальну та тестову вибірки без використання готових функцій, таких як `train_test_split`. Ручне розділення допомагає краще зрозуміти основи обробки даних перед побудовою моделі.

a. Проаналізуйте приклад для набору Iris:

```
import numpy as np
from sklearn.datasets import load_iris
```

```
iris = load\_iris()
data = iris.data
target = iris.target

np.random.seed(42)

# Створюємо одновимірний масив, що містить
# послідовність чисел від 0 до (кількість зразків - 1).
indices = np.arange(data.shape[0])

# Переміщуємо індекси для зразків у випадковому порядку
np.random.shuffle(indices)

# Встановимо розмір тренувальної вибірки 70% даних
train_size = int(0.7 * len(indices))

# Розділяємо індекси
train_indices = indices[:train_size]
test_indices = indices[train_size:]

# Створення тренувальних і тестових наборів даних

X_train = data[train_indices]
y_train = target[train_indices]
X_test = data[test_indices]
y_test = target[test_indices]

# Перевіряємо розміри вибірок
print("Розмір тренувального набору:", X\_train.shape)
print("Розмір тестового набору:", X\_test.shape)
```

b. Розділіть дані на тренувальну та тестову вибірки для набору Titanic.