МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Кафедра інформаційних систем в економіці

ЗВІТ

з лабораторної роботи № 2

з дисципліни «Системи і методи штучного інтелекту»

Тема: “ ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ”

Виконала:
студентка групи ІН-402
Пушенко Я. О.
Перевірив:
Волошин А. П.

Київ 2025

Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити різні методи класифікації даних та навчитися їх порівнювати.

Посилання на github : https://github.com/YanaPushenko/AI?tab=readme-ov-file#ai

Завдання 2.1. Класифікація за допомогою машин опорних векторів (SVM)

У межах завдання було реалізовано класифікацію даних за допомогою методу машин опорних векторів (SVM). Основною метою було створення класифікатора, здатного прогнозувати, чи перевищує річний дохід певної особи $50 000 на основі її соціально-демографічних характеристик. Для цього використовувався відкритий набір даних "Adult Income Dataset", що поєднує числові та категоріальні ознаки.

Імпортуємо бібліотеки

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import train_test_split, cross_val_score
```

Вхідний файл

```
[9]: # Вхідний файл, який містить дані
     input_file = 'income_data.txt'
```

Читання даних

```
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000
```

Здійснюємо зчитування даних із файлу income_data.txt, виконує їх первинну очистку та формує збалансований набір для класифікації. Під час проходження по кожному рядку кодується умова зупинки після досягнення 25 000 прикладів для кожного з двох класів (<=50K та >50K). Усі записи, що містять пропущені значення, позначені символом ?, ігноруються. Рядки розбиваються на окремі атрибути за допомогою роздільника кома, після чого, залежно від класу, додаються до масиву вхідних даних.

```
with open(input_file, 'r') as f:
    for line in f.readlines():
        if count_class1 >= max_datapoints and count_class2 >= max_datapoints:
            break

        if '?' in line:
            continue

        data = line[:-1].split(',')
        # print(data)

        if data[-1] == ' <=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1

        if data[-1] == ' >50K' and count_class2 < max_datapoints:
            X.append(data)
            count_class2 += 1
```

Перетворюємо всі текстові (категоріальні) ознаки у числові за допомогою LabelEncoder, що є необхідним етапом перед подачею даних до машинного алгоритму. Кожен нечисловий стовпець кодується окремо. В результаті формується матриця ознак X та цільовий вектор y, готові для навчання моделі.

```
X = np.array(X)
```

```
# Перетворення рядкових даних на числові
label_encoder = []
X_encoded = np.empty(X.shape)
for i, item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])

X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)
```

```
classifier = OneVsRestClassifier(LinearSVC(random_state=0))
```

Створюємо та навчає класифікатор SVM із використанням стратегії "один проти решти" (OneVsRestClassifier) і лінійного ядра (LinearSVC). Після первинного навчання на всіх даних відбувається поділ на навчальну і тестову вибірки у співвідношенні 80/20. Модель повторно навчається на навчальних даних і виконує прогнозування на тестовому наборі. Для оцінки якості класифікації обчислюється середнє значення метрики F1 із використанням крос-валідації (3 фолди).

```
classifier = OneVsRestClassifier(LinearSVC(random_state=0))
```

```
classifier.fit(X, y)
```

```
[22]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)

[23]: classifier = OneVsRestClassifier(LinearSVC(random_state=0))
      classifier.fit(X_train, y_train)
      y_test_pred = classifier.predict(X_test)
```

```
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print('F1 score: ' + str(round(100*f1.mean(), 2)) + '%')
```

```
: input_data = ['37', ' Private', ' 215646', ' HS-grad', ' 9', ' Never-married', ' Handler
```

Кожен елемент input_data перетворюється у числове представлення: якщо значення є числом — воно перетворюється без змін, якщо ж це текст — застосовується відповідний LabelEncoder, збережений раніше під час обробки основного набору даних. Закодований масив перетворюється у формат, придатний для класифікатора, після чого виконується прогноз, і результат виводиться у первісному (текстовому) вигляді.

```
: input_data_encoded = [-1] * len(input_data)
  count = 0
  for i, item in enumerate(input_data):
      if item.isdigit():
          input_data_encoded[i] = int(item)
      else:
          input_data_encoded[i] = label_encoder[count].transform([item])[0]
          count += 1

  input_data_encoded = np.array(input_data_encoded).reshape(1, -1)
```

```
: predicted_class = classifier.predict(input_data_encoded)
```

```
: print(label_encoder[-1].inverse_transform(predicted_class)[0])
```

```
 >50K
```

Цей код розраховує основні метрики якості класифікації: акуратність (accuracy), точність (precision), повноту (recall) та F1-міру. Усі показники обчислюються з використанням зваженого середнього (average='weighted'), що враховує дисбаланс класів. Результати виводяться у відсотках і дозволяють оцінити ефективність навченої моделі на тестовій вибірці.

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
#Оцінення значення інших показників якості класифікації
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, average='weighted')
recall = recall_score(y_test, y_test_pred, average='weighted')
f1 = f1_score(y_test, y_test_pred, average='weighted')

print(f'Акуратність: {accuracy*100:.2f}%')
print(f'Точність: {precision*100:.2f}%')
print(f'Повноті: {recall*100:.2f}%')
print(f'F1: {f1*100:.2f}%')
```

```
Акуратність: 29.42%
Точність: 70.98%
Повноті: 29.42%
F1: 19.14%
```

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами та

Завдання 2.4. Порівняння якості класифікаторів для набору даних

Підключаємо бібліотеки, завантажуємо дані з файлу 'income_data.txt', обробляючи кожен рядок, щоб витягнути інформацію про ознаки (X) та класи (y) для двокласової задачі класифікації (<=50K та >50K). Якщо рядок містить '?', він ігнорується. Дані для кожного класу обмежені значенням max_datapoints, тобто по 25,000 записів для кожного класу. Після цього дані зберігаються в масиві X, готовому для подальшої обробки чи моделювання.

```python
# Підключення бібліотек
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import LinearSVC, SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import train_test_split, cross_val_score
```

```python
# Вхідний файл, який містить дані
input_file = 'income_data.txt'
```

```python
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000
```

```python
with open(input_file, 'r') as f:
    for line in f.readlines():
        if count_class1 >= max_datapoints and count_class2 >= max_datapoints:
            break

        if '?' in line:
            continue

        data = line[:-1].split(',')
        # print(data)

        if data[-1] == ' <=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1

        if data[-1] == ' >50K' and count_class2 < max_datapoints:
            X.append(data)
            count_class2 += 1
```

```python
X = np.array(X)
```

Виконуємо кодування категоріальних змінних у числовий формат за допомогою LabelEncoder, створюючи нову матрицю X_encoded, де кожен стовпець перетворюється на числові значення. Далі, для кожного класифікатора SVM (RBF, лінійного, поліноміального з степенем 3 і сигмоїдального) за допомогою OneVsRestClassifier проводиться навчання та тестування моделі на вибірці даних. Для кожного класифікатора обчислюються та виводяться метрики: акуратність, точність, повнота та F1-міра. Це дозволяє оцінити ефективність різних моделей SVM на даному наборі даних, де кожна метрика розраховується для зваженого середнього значення по класах.

```
label_encoder = []
X_encoded = np.empty(X.shape)
for i, item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])

X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)
```

```
categories = {
    'RBF SVM': SVC(kernel='rbf', random_state=0),
    'Linear SVM': LinearSVC(random_state=0),
    'Poly SVM (degree=3)': SVC(kernel='poly', degree=3, random_state=0),
    'Sigmoid SVM': SVC(kernel='sigmoid', random_state=0)
}
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
for categor in categories.items():
    print(f'\nКласифікатори SVM: {categor[0]}')
    clf = OneVsRestClassifier(categor[1])
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    clf.fit(X_train, y_train)
    y_test_pred = clf.predict(X_test)

    accuracy = accuracy_score(y_test, y_test_pred)
    precision = precision_score(y_test, y_test_pred, average='weighted', zero_division=0)
    recall = recall_score(y_test, y_test_pred, average='weighted')
    f1 = f1_score(y_test, y_test_pred, average='weighted')

    print(f'Акуратність: {accuracy*100:.2f}%')
    print(f'Точність: {precision*100:.2f}%')
    print(f'Повнота: {recall*100:.2f}%')
    print(f'F1: {f1*100:.2f}%')
    print('-' * 40)
```

```
Класифікатори SVM: RBF SVM
Акуратність: 74.64%
Точність: 55.71%
Повнота: 74.64%
F1: 63.80%
----------------------------------------

Класифікатори SVM: Linear SVM
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:31: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in
1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\svm\_base.py:1237: ConvergenceWarning: Liblinear failed to converge, increase the number of iteration
s.
  warnings.warn(
Акуратність: 78.30%
Точність: 77.13%
Повнота: 78.30%
F1: 73.93%
----------------------------------------

Класифікатори SVM: Poly SVM (degree=3)
Акуратність: 74.64%
Точність: 55.71%
Повнота: 74.64%
F1: 63.80%
----------------------------------------

Класифікатори SVM: Sigmoid SVM
Акуратність: 63.82%
Точність: 63.56%
Повнота: 63.82%
F1: 63.68%
----------------------------------------
```

Далі використовуємо раніше навчені енкодери для перетворення вхідних

даних input_data у формат, зручний для подачі до моделей SVM. Кожен елемент списку перетворюється на числове значення або за допомогою LabelEncoder для категоріальних змінних, або безпосередньо для числових значень. Потім, для кожного класифікатора SVM з попередньо визначеного набору моделей, виконується навчання на всіх даних і робиться передбачення класу для нових вхідних даних. Результат передбаченого класу відображається після інвертування його з числового формату назад у вихідну категорію за допомогою останнього енкодера.

```
input_data = ['37', ' Private', ' 215646', ' HS-grad', ' 9', ' Never-married', ' Handlers-cleaners', ' Not-in-family', ' White', ' Male', ' 0', ' 0', '
```

```
input_data_encoded = [-1] * len(input_data)
count = 0
for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = int(item)
    else:
        input_data_encoded[i] = label_encoder[count].transform([item])[0]
        count += 1

input_data_encoded = np.array(input_data_encoded).reshape(1, -1)
```

```
for categor in categories.values():
    print(f'\nКласифікатори SVM: {categor.__class__.__name__}')
    categor.fit(X, y)
    predicted_class = categor.predict(input_data_encoded)
    print(label_encoder[-1].inverse_transform(predicted_class)[0])
    print('-' * 40)
```

```
Класифікатори SVM: SVC
 <=50K
----------------------------------------

Класифікатори SVM: LinearSVC
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\svm
1.5. Set the value of `dual` explicitly to suppress th
  warnings.warn(
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\svm
s.
  warnings.warn(
 <=50K
----------------------------------------

Класифікатори SVM: SVC
 <=50K
----------------------------------------

Класифікатори SVM: SVC
 <=50K
----------------------------------------
```

Імпортуємо необхідні бібліотеки для візуалізації результатів та моделювання, а також створює словник з різними класифікаторами, що будуть використовуватися для тренування та оцінки ефективності. Моделі включають логістичну регресію (через OneVsRestClassifier), дерево рішень, алгоритм найближчих сусідів, лінійний дискримінантний аналіз і наївний байєсівський класифікатор. Результати кожної моделі будуть зберігатися в списку results, а імена моделей — у списку names

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
```

```python
# Список моделей
models = {
    'Logistic Regression': OneVsRestClassifier(LogisticRegression(solver='liblinear')),
    'Decision Tree': DecisionTreeClassifier(random_state=0),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Linear Discriminant Analysis': LinearDiscriminantAnalysis(),
    'Gaussian Naive Bayes': GaussianNB(),
}

results = []
names = []
```

Крос-валідація і збір результатів

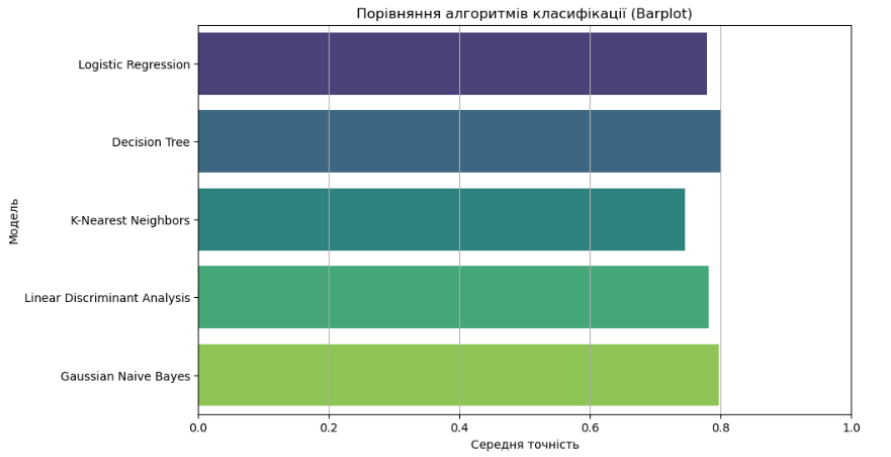
```python
# Крос-валідація і збір результатів
for name, model in models.items():
    kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results.mean())
    names.append(name)
    print(f"{name}: {cv_results.mean():.4f} (+/- {cv_results.std():.4f})")
```

```
Logistic Regression: 0.7789 (+/- 0.0083)
Decision Tree: 0.7992 (+/- 0.0085)
K-Nearest Neighbors: 0.7455 (+/- 0.0075)
Linear Discriminant Analysis: 0.7809 (+/- 0.0060)
Gaussian Naive Bayes: 0.7971 (+/- 0.0060)
```

Візуалізований результат

```
# Візуалізація результатів
plt.figure(figsize=(10, 6))
sns.barplot(x=results, y=names, palette="viridis")
plt.xlabel('Середня точність')
plt.ylabel('Модель')
plt.title('Порівняння алгоритмів класифікації (Barplot)')
plt.xlim(0, 1)
plt.grid(axis='x')
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_5180\652785093.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for
same effect.

  sns.barplot(x=results, y=names, palette="viridis")
```



З результатів тренування для різних моделей видно, що **Decision Tree** має найвищу середню точність (0.7992), що робить її найкращою серед цих моделей за точністю на даному наборі даних. Водночас, варто звернути увагу на те, що моделі **Gaussian Naive Bayes** (0.7971) і **Logistic Regression** (0.7789) демонструють також непогані результати, хоча їх точність трохи нижча, ніж у дерева рішень.

Моделі **K-Nearest Neighbors** (0.7455) і **Linear Discriminant Analysis** (0.7809) займають нижчі позиції за результатами точності, але все одно можуть бути корисними в залежності від специфіки задачі, особливо для великих чи складних наборів даних.

Таким чином, з огляду на ці результати, **Decision Tree** є найкращою моделлю для класифікації в рамках цього експерименту.

Завдання 2.3. Порівняння якості класифікаторів на прикладі класифікації сортів ірисів

1)

Підключаємо iris з бібліотеки sklearn.datasets, що містить інформацію про три види ірисів. Після цього виводимо список ключів, які містяться в цьому наборі даних.

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

```
print('Ключі iris dataset: \n{}'.format(iris_dataset.keys()))
```

```
Ключі iris dataset:
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

Виводимо кілька важливих аспектів набору даних iris_dataset. Спочатку код показує частину опису набору даних (перші 193 символи) і додає трикрапки, щоб вказати на наявність більшого опису. Далі виводяться:

- Назви відповідей (target_names): це назви класів (видів квітів).
- Назви ознак (feature_names): це характеристики, які були виміряні для кожної квітки (довжина та ширина чашолистка і пелюстки).
- Тип масиву data: тип даних, який зберігається в масиві (ймовірно, це буде numpy.ndarray).
- Форма масиву data: розміри масиву, який містить самі дані, де кількість рядків — це кількість зразків, а стовпців — кількість ознак для кожного зразка.

```
print(iris_dataset['DESCR'][:193] + "\n...")
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive
...
```

```
print('Назви відповідей: {}'.format(iris_dataset['target_names']))
```

```
Назви відповідей: ['setosa' 'versicolor' 'virginica']
```

```
print('Назви ознак: {}'.format(iris_dataset['feature_names']))
```

```
Назви ознак: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

```
print('Тип масиву data: {}'.format(type(iris_dataset['data'])))
```

```
Тип масиву data: <class 'numpy.ndarray'>
```

```
print('Форма масиву data: {}'.format(iris_dataset['data'].shape))
```

```
Форма масиву data: (150, 4)
```

Перші п’ять записів

```
[15]: print(f"Перші п’ять записів у масиві data:\n{iris_dataset['data'][:5]}")
```

```
Перші п’ять записів у масиві data:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

Цей код виводить два важливих аспекти набору даних. Спочатку він показує тип масиву target, який містить мітки класів для кожного зразка, і виведе, що це масив типу numpy.ndarray. Потім він виводить самі цільові значення (відповіді), які є числами, що представляють різні види ірисів у наборі даних

```
print('Тип масиву target: {}'.format(type(iris_dataset['target'])))
```

```
Тип масиву target: <class 'numpy.ndarray'>
```

```
print('Відповіді: {}'.format(iris_dataset['target']))
```

```
Відповіді: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

2)

Завантажуємо набір даних ірисів з URL-адреси за допомогою бібліотеки pandas, використовуючи функцію read_csv(). Після цього встановили назви стовпців для цього набору даних: sepal-length, sepal-width, petal-length, petal-width та class (для міток класів, що відповідають видам ірисів). Використовуємо метод shape, щоб вивести розміри набору даних (кількість рядків та стовпців). Також за допомогою методу head(20) виводимо перші 20 рядків набору даних, щоб ознайомитися з його структурою та вмістом.

```
import pandas as pd
import numpy as np
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

```
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pd.read_csv(url, names=names)
```

```
print(dataset.shape)
```

```
(150, 5)
```

```
dataset.head(20)
```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa

Статистичні показники для кожної ознаки. Далі групуємо дані за класами (видами ірисів) і виводимо кількість зразків для кожного виду за допомогою groupby('class').size(). Потім створюємо графіки типу "boxplot" для кожної з числових ознак, щоб візуально оцінити їх розподіл і виявити можливі викиди. Наприкінці, за допомогою plt.show(), виводимо ці графіки на екран.

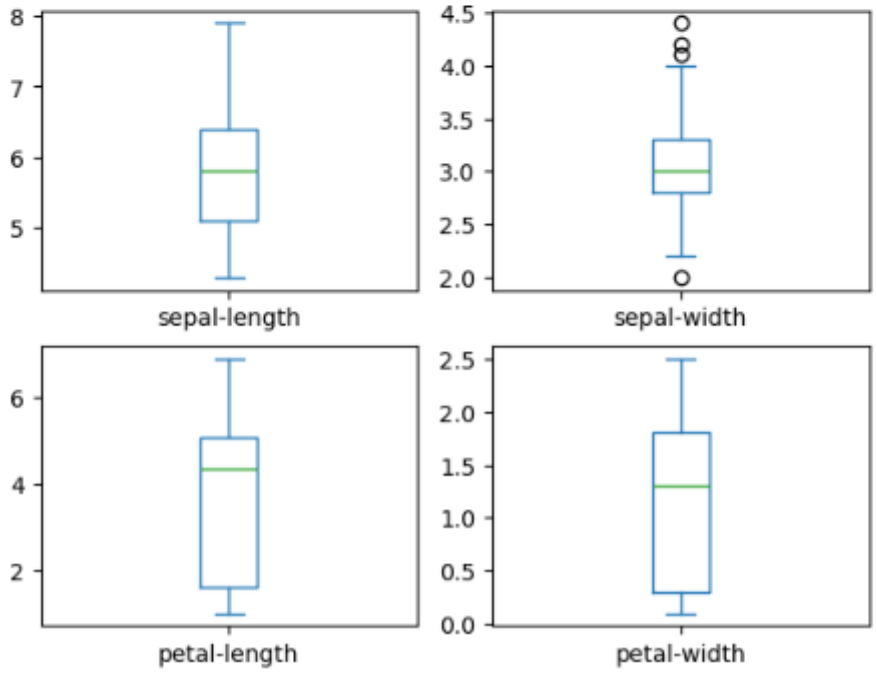
```
dataset.describe()
```

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
print(dataset.groupby('class').size())
```

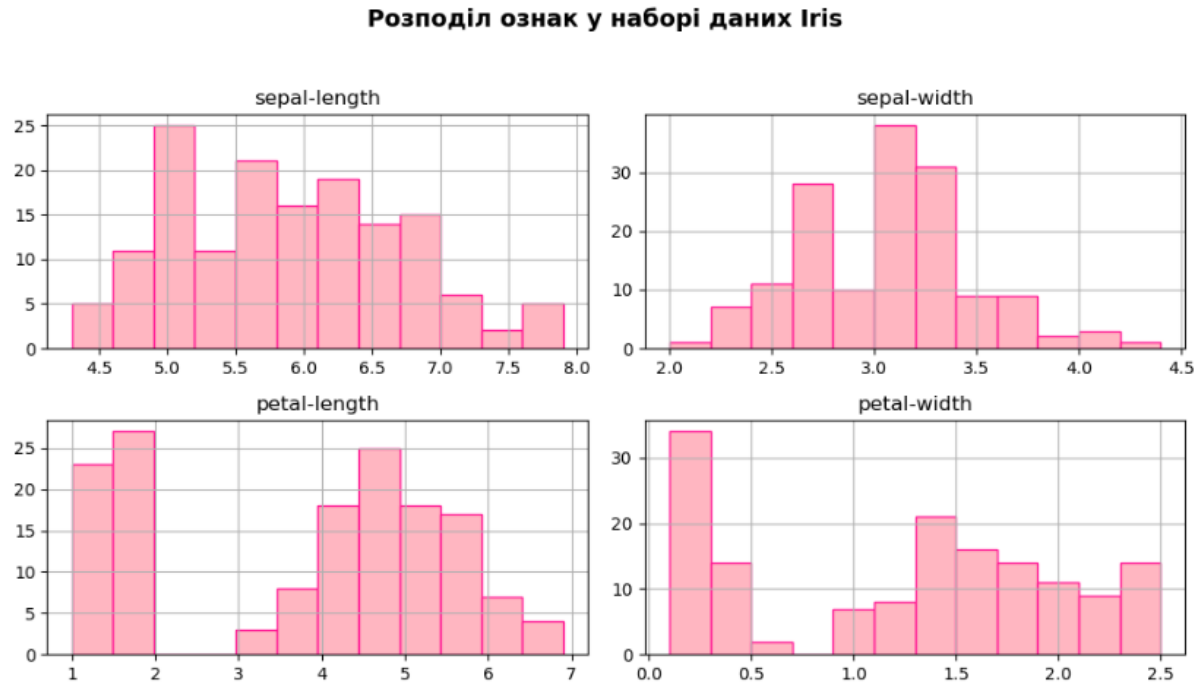
```
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

```
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
```



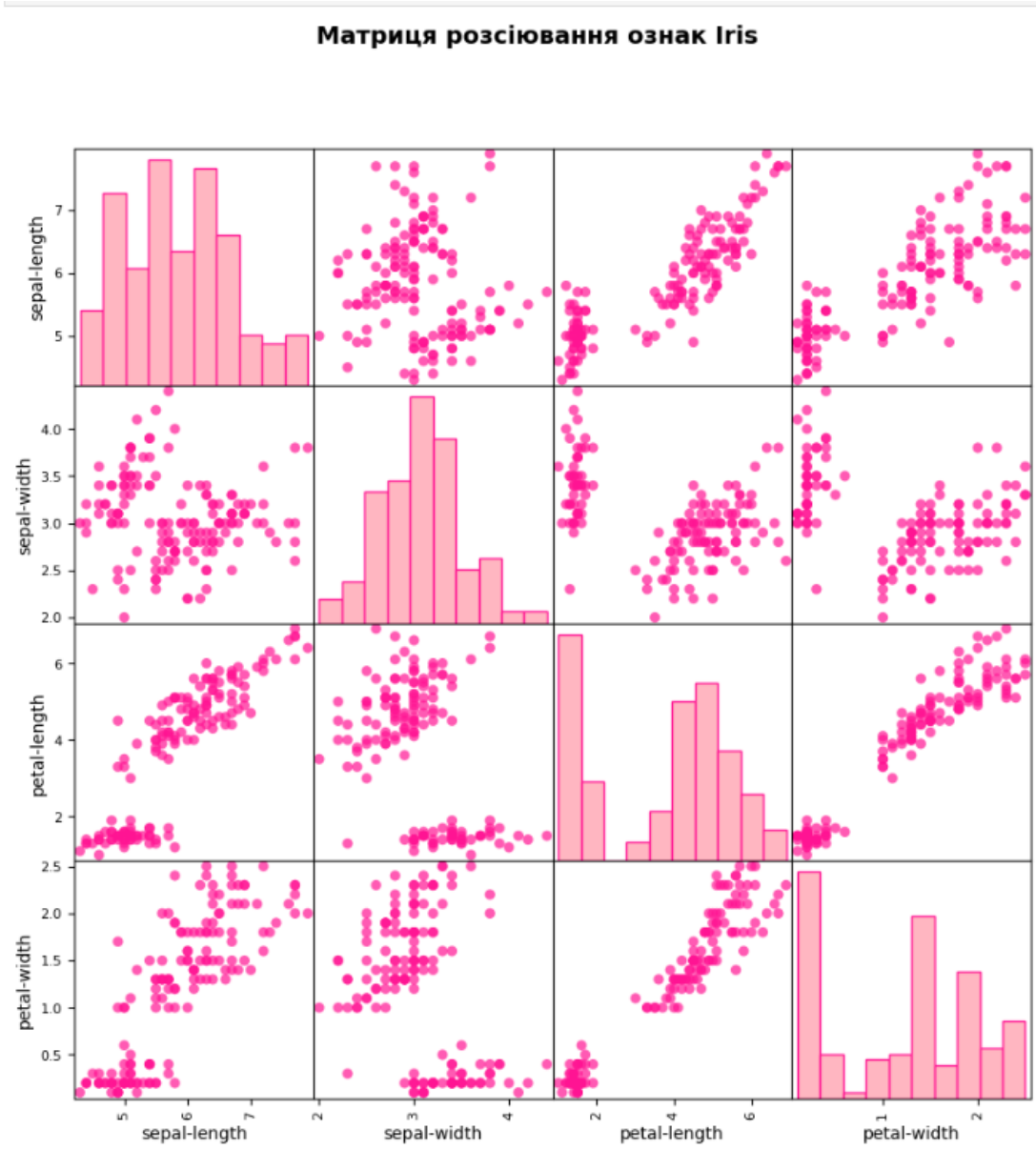
Гістограма

```
dataset.hist(
    bins=12,
    figsize=(10, 6),
    color='lightpink',
    edgecolor='deeppink'
)

plt.suptitle('Розподіл ознак у наборі даних Iris', fontsize=14, fontweight='bold')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # для кращого розміщення заголовка
plt.show()
```



Матриця розсіювання ознак

```
scatter_matrix(
    dataset,
    figsize=(10, 10),
    alpha=0.7,
    diagonal='hist',
    color='deeppink',
    marker='o',
    hist_kwds={'color': 'lightpink', 'edgecolor': 'deeppink'}
)

plt.suptitle("Матриця розсіювання ознак Iris", fontsize=14, fontweight='bold')
plt.show()
```



Розділяємо дані на ознаки та мітки, де в X зберігаються перші чотири стовпці (ознаки), а в y — останній стовпець (мітки класів). Потім за допомогою функції train_test_split() розбиваємо дані на навчальну та тестову вибірки, де тестова вибірка становить 20% від усіх даних. Після цього надано результат точності для різних моделей класифікації, таких як логістична регресія (LR), лінійний дискримінантний аналіз (LDA), K найближчих сусідів (KNN), дерево рішень (CART), наївний байєсівський класифікатор (NB) та опорні вектори (SVM). Найвищу точність показала модель SVM (98.33%), що вказує на її найкращу ефективність серед усіх використаних методів.

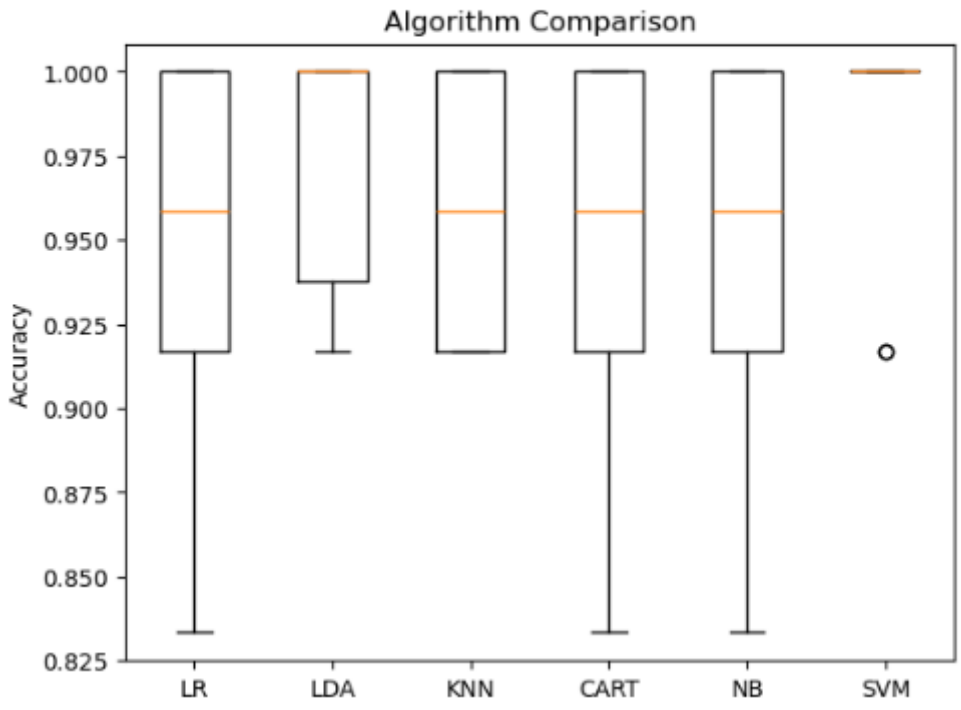
```
array = dataset.values
X = array[:,0:4]
y = array[:,4]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

```
models = []
models.append(('LR', OneVsRestClassifier(LogisticRegression(solver='liblinear'))))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

results = []
names = []

for name, model in models:
    kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.941667 (0.065085)
LDA: 0.975000 (0.038188)
KNN: 0.958333 (0.041667)
CART: 0.950000 (0.055277)
NB: 0.950000 (0.055277)
SVM: 0.983333 (0.033333)
```

Графік

```
#  Порівняння алгоритмів
plt.boxplot(results, labels=names)
plt.title('Algorithm Comparison')
plt.ylabel('Accuracy')
plt.show()
```



Тут ми тренуємо модель SVM на навчальних даних та робимо прогноз на тестовій вибірці, використовуючи метод predict(). Оцінюємо ефективність моделі за допомогою точності (accuracy_score), матриці сплутаних класів (confusion_matrix) та звіту про класифікацію (classification_report). Модель показала точність 96.67%, з хорошими результатами для кожного з видів ірисів. Далі ми створюємо нові зразки даних і робимо прогноз для них, де модель передбачила, що перший зразок належить до Iris-setosa, а другий — до Iris-versicolor.

```
#  Створюємо прогноз на контрольній вибірці
model = SVC(gamma='auto')
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

```
# Оцінюємо прогноз
print(accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
0.9666666666666667
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      0.92      0.96        13
 Iris-virginica       0.86      1.00      0.92         6

       accuracy                           0.97        30
      macro avg       0.95      0.97      0.96        30
   weighted avg       0.97      0.97      0.97        30
```

```
нові_зразки = [
    [5.5, 3.2, 1.4, 0.2],
    [5.7, 3.1, 4.0, 1.1]
]

X_new = np.array(нові_зразки)

print(f'Розмірність масиву X_new: {X_new.shape}')
```

```
Розмірність масиву X_new: (2, 4)
```

```
predictions = model.predict(X_new)
print(f'Прогноз для X_new: {predictions}')
```

```
Прогноз для X_new: ['Iris-setosa' 'Iris-versicolor']
```

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

Завантажуємо набір даних Iris за допомогою функції load_iris() і зберігаємо ознаки в X, а мітки класів у y.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import RidgeClassifier
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from io import BytesIO
```

```
# Завантаження даних Iris
iris = load_iris()
X, y = iris.data, iris.target
```

Розбиваємо набір даних Iris на тренувальну та тестову вибірки, потім тренуємо модель **RidgeClassifier** з параметром tol=1e-2 та методом розв'язування "sag". Після цього робимо прогнози на тестовій вибірці. Оцінюємо модель за кількома метриками: точністю (accuracy), точністю за класами (precision), відтворюваністю (recall), F1-мірою, коефіцієнтом Коена та коефіцієнтом кореляції Метьюса. Модель показала точність 75.56%, з високою точністю для класів та середніми результатами для інших метрик.

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.3, random_state=0)
```

```
clf = RidgeClassifier(tol=1e-2, solver="sag")
clf.fit(Xtrain, ytrain)
```

```
RidgeClassifier
RidgeClassifier(solver='sag', tol=0.01)
```

```
ypred = clf.predict(Xtest)
```

```
# Оцінка якості моделі
print('Accuracy:', np.round(metrics.accuracy_score(ytest, ypred), 4))
print('Precision:', np.round(metrics.precision_score(ytest, ypred, average='weighted'), 4))
print('Recall:', np.round(metrics.recall_score(ytest, ypred, average='weighted'), 4))
print('F1 Score:', np.round(metrics.f1_score(ytest, ypred, average='weighted'), 4))
print('Cohen Kappa Score:', np.round(metrics.cohen_kappa_score(ytest, ypred), 4))
print('Matthews Corrcoef:', np.round(metrics.matthews_corrcoef(ytest, ypred), 4))
```

```
Accuracy: 0.7556
Precision: 0.8333
Recall: 0.7556
F1 Score: 0.7503
Cohen Kappa Score: 0.6431
Matthews Corrcoef: 0.6831
```

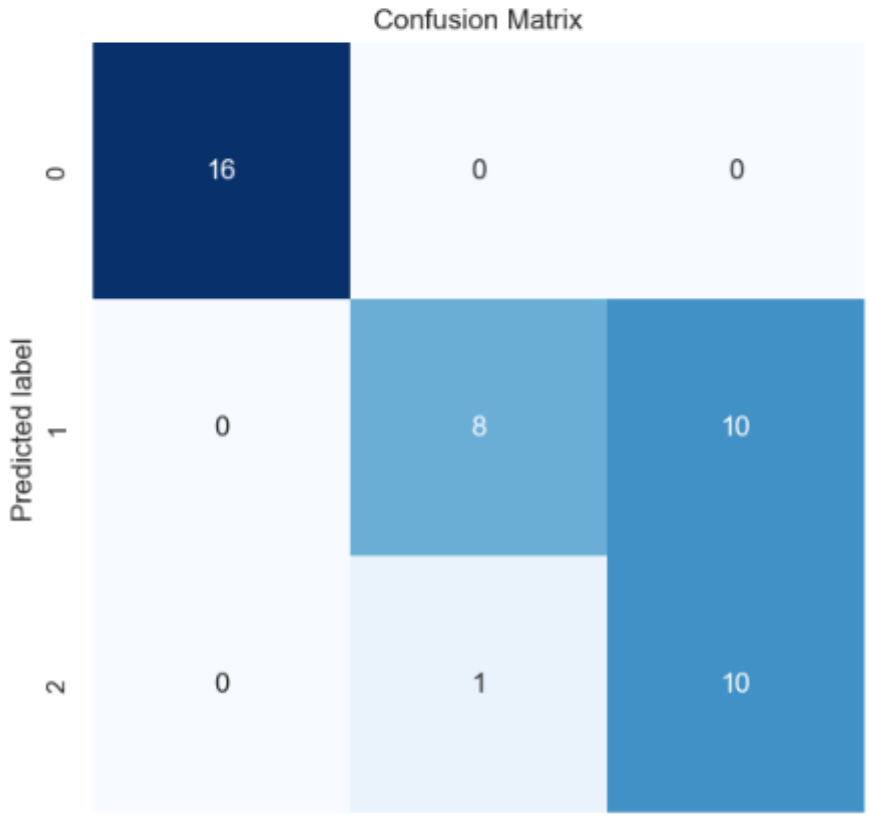
Звіт про класифікацію для нашої моделі, що включає точність, відтворюваність та F1-міру для кожного класу, а також середні значення для всіх класів. Звіт показує, що модель добре класифікує перший клас (точність та відтворюваність 1.00), але має менші показники для других двох класів. Потім ми створюємо матрицю сплутаних класів за допомогою confusion_matrix() і виводимо її у вигляді теплової карти за допомогою бібліотеки seaborn

```
print('\nClassification Report:\n', metrics.classification_report(ytest, ypred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.89      0.44      0.59        18
           2       0.50      0.91      0.65        11

    accuracy                           0.76        45
   macro avg       0.80      0.78      0.75        45
weighted avg       0.83      0.76      0.75        45
```

```
mat = confusion_matrix(ytest, ypred)
```

```
sns.set()
plt.figure(figsize=(7, 6))
sns.heatmap(mat, annot=True, fmt='d', cmap="Blues", cbar=False, square=True)
plt.xlabel('True label')
plt.ylabel('Predicted label')
plt.title('Confusion Matrix')
plt.savefig("Confusion.jpg")
```



Налаштування класифікатора **RidgeClassifier** включають параметр **tol=1e-2**, який визначає поріг для зупинки алгоритму. Якщо зміна в коефіцієнтах моделі стає меншою за 0.01, алгоритм припиняє навчання. Інший параметр, **solver="sag"**, вибирає метод оптимізації для пошуку мінімуму функції втрат. "SAG" (Stochastic Average Gradient) є стохастичним методом, що дозволяє швидше досягати оптимальних значень для великих наборів даних.

Оцінка якості моделі базується на кількох показниках. **Accuracy** або точність показує, який відсоток правильно класифікованих випадків серед усіх передбачених. У нашому випадку це **75.56%**. **Precision** (точність) вимірює, скільки з передбачених позитивних випадків є дійсно позитивними. Для класу 0 точність дорівнює **1.00**, для класу 1 — **0.89**, а для класу 2 — **0.50**. **Recall** (відтворюваність) вимірює, скільки з усіх дійсно позитивних випадків модель правильно класифікувала як позитивні. Для класу 0 це **1.00**, для класу 1 — **0.44**, для класу 2 — **0.91**. **F1 Score** комбінує точність і відтворюваність, даючи збалансовану міру. У нашому випадку для класу 0 це **1.00**, для класу 1 — **0.59**, для класу 2 — **0.65**. Також використовуються **Cohen Kappa Score** і **Matthews Correlation Coefficient (MCC)**, які оцінюють узгодженість між передбаченнями та реальними класами, з урахуванням ймовірних помилок.

Матриця сплутаних класів, представлена на зображенні "Confusion.jpg", показує, скільки з кожного класу було правильно чи неправильно класифіковано. У нашому випадку клас 0 був класифікований без помилок, для класу 1 модель допустила одну помилку, а для класу 2 було кілька помилок.

Коефіцієнт Коена Каппа вимірює узгодженість між двома оцінювачами або класифікаторами, з урахуванням випадкових помилок. Значення 0.6431 свідчить про помірну узгодженість. Коефіцієнт кореляції Метьюза (MCC) враховує всі типи помилок (позитивні, негативні) та є збалансованою метрикою, що показує загальну ефективність класифікації. У нашому випадку значення 0.6831 вказує на добру загальну ефективність моделі.

Висновок: В результаті проведеного дослідження з використанням спеціалізованих бібліотек Python, таких як scikit-learn, були вивчені різні методи класифікації даних, зокрема RidgeClassifier, SVM, Logistic Regression, K-Nearest Neighbors, Decision Tree та інші. Завдяки інструментам Python ми змогли не лише застосувати різноманітні алгоритми до реальних даних, а й оцінити їх ефективність за допомогою кількох важливих показників якості, таких як accuracy, precision, recall, F1 score, Cohen's Kappa та Matthews Correlation Coefficient. Ці метрики дозволяють порівнювати не лише загальну точність моделей, а й їхню здатність правильно класифікувати кожен з класів, а також справлятися з різними видами помилок.

У результаті порівняння моделей класифікації для набору даних Iris можна зробити висновок, що найкращим результатом за точністю та іншими показниками продемонстрував SVM. Однак, для різних наборів даних та типів задач, результат може варіюватися, що підкреслює важливість вибору підходящої моделі в залежності від конкретної ситуації.

Отже, використання Python і бібліотек для машинного навчання дозволяє не лише глибше зрозуміти принципи класифікації, а й ефективно застосовувати

різні алгоритми для порівняння результатів та оптимізації моделі для конкретних завдань.