ЛАБОРАТОРНА РОБОТА № 2

Тема: *«***ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ***»*

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

Хід роботи

Посилання на програмнй код на Github: https://github.com/NightSDay/AI-2023/tree/main/AI-subject/L-2_Comparison

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

Створіть класифікатор у вигляді машини опорних векторів, призначений для прогнозування меж доходу заданої фізичної особи на основі 14 ознак (атрибутів). Метою є з'ясування умов, за яких щорічний прибуток людини перевищує $50000 або менше цієї величини за допомогою бінарної класифікації.

Випишіть у звіт всі 14 ознак з набору даних – їх назви, що вони позначають та вид (числові чи категоріальні).

Обчисліть значення інших показників якості класифікації (акуратність, повнота, точність) та разом з F1 занесіть їх у звіт. (Див. ЛР-1).

Зробіть висновок до якого класу належить тестова точка.

14 ознак з набору даних

1. **age (вік)**: Це числове поле, яке представляє вік особи.
2. **workclass (клас роботи)**: Це категоріальне поле, яке вказує на тип зайнятості особи. Наприклад: приватний сектор, самозайнята особа, державний сектор, безробітний тощо.

3. **fnlwgt**: Це числове поле.
4. **education (освіта)**: Це категоріальне поле, яке вказує на рівень освіти особи. Наприклад: бакалавр, загальна середня освіта, магістр, доктор наук тощо.
5. **education-num (рівень освіти у числовому форматі)**: Це числове полеЦ.
6. **marital-status (сімейний стан)**: Це категоріальне поле, яке вказує на сімейний стан особи. Наприклад: одружений/одружена, розлучений/розлучена, ніколи не одружений/одружена тощо.
7. **occupation (професія)**: Це категоріальне поле, яке вказує на рід діяльності або професійну область особи. Наприклад: інформаційні технології, медична сфера, продажі тощо.
8. **relationship (відносини)**: Це категоріальне поле, яке вказує на відносини особи в сім'ї. Наприклад: дружина, дитина, чоловік, інша родичка тощо.
9. **race (раса)**: Це категоріальне поле, яке вказує на расу чи етнічну приналежність особи. Наприклад: білий, азіатсько-тихоокеанський, американський індіанець тощо.
10. **sex (стать)**: Це категоріальне поле, яке вказує на стать особи.
11. **capital-gain (приріст капіталу)**: Це числове поле, яке може представляти суму приросту капіталу.
12. **capital-loss (втрати капіталу)**: Це числове поле, яке може представляти суму втрат капіталу.
13. **hours-per-week (годин на тиждень)**: Це числове поле, яке вказує кількість годин, які особа працює на тиждень.
14. **native-country (країна походження)**: Це категоріальне поле, яке вказує на країну, з якої особа походить.

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn.model_selection import train_test_split, cross_val_score

input_file = 'income_data.txt'
```

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

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(', ')

        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

X = np.array(X)

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)

scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
X = scaler.fit_transform(X)

classifier = OneVsOneClassifier(LinearSVC(random_state=0, dual=True))

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran-
dom_state=5)

scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
X_train = scaler.fit_transform(X_train)


classifier.fit(X=X_train, y=y_train)
y_test_pred = classifier.predict(X_test)

f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy', cv=3)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
precision_values = cross_val_score(classifier, X, y, scoring='precision_weighted',
cv=3)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
recall_values = cross_val_score(classifier, X, y, scoring='recall_weighted', cv=3)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")
```

```
f1_values = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")

print("F1 score: " + str(round(100*f1.mean(), 2)) + "%")

input_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married', 'Han-
dlers-cleaners', 'Not-in-family', 'White', 'Male', '0', '0', '40', 'United-
States']

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


predicted_class = classifier.predict(input_data_encoded)
print(label_encoder[-1].inverse_transform(predicted_class)[0])
```

Результат виконання програми:

```
PS E:\AI-subject\L-2 Comparison> python LR_2_task_1.py
Accuracy: 81.95%
Precision: 80.94%
Recall: 81.95%
F1: 80.13%
F1 score: 80.13%
```

Рис. 1. Результат виконання програми

Висновок, щодо того до якого класу належить тестова точка

Згідно з результатами виконання програми, тестова точка належить до класу >50K, тобто людина заробляє більше 50К.

Завдання 2.2. **Порівняння якості класифікаторів SVM з нелінійними ядрами**

У попередньому завданні ми побачили, як простий алгоритм SVM LinearSVC може бути використаний для знаходження межі рішення для лінійних даних. Однак у разі нелінійно розділених даних, пряма лінія не може бути використана як межа прийняття рішення. Натомість використовується модифікована версія SVM, звана Kernel SVM. В основному, ядро SVM проектує дані нижніх

вимірювань, що нелінійно розділяються, на такі, що лінійно розділяються більш високих вимірювань таким чином, що точки даних, що належать до різних класів, розподіляються за різними вимірами. В цьому є закладена складна математика, але вам не потрібно турбуватися про це, щоб використовувати SVM. Ми можемо просто використовувати бібліотеку Scikit-Learn Python для реалізації та використання SVM ядра. Реалізація SVM ядра за допомогою Scikit-Learn аналогічна до простого SVM.

Використовуючи набір даних та код з попереднього завдання створіть та дослідіть нелінійні класифікатори SVM.

з поліноміальним ядром;

з гаусовим ядром;

з сигмоїдальним ядром.

Для кожного виду класифікатора отримайте та запишіть у звіт показники якості алгоритму класифікації.

Лістинг оголошення класифікатора SVM з поліноміальним ядром:

```
classifier = OneVsOneClassifier(SVC(kernel='poly', degree=8, random_state=0))
```

Результат виконання програми (класифікатор SVM з поліноміальним ядром):

```
Accuracy: 83.75%
Precision: 83.06%
Recall: 83.75%
F1: 83.2%
<=50K
```

Рис. 2.1. Результат виконання програми (класифікатор SVM з поліноміальним ядром)

Лістинг оголошення класифікатора SVM з гаусовим ядром:

```
classifier = OneVsOneClassifier(SVC(kernel='rbf'))
```

Результат виконання програми (класифікатор SVM з гаусовим ядром):

```
Accuracy: 83.96%
Precision: 83.18%
Recall: 83.96%
F1: 82.95%
<=50K
```

Рис. 2.2. Результат виконання програми (класифікатор SVM з гаусовим ядром)

Лістинг оголошення класифікатора SVM з сигмоїдальним ядром:

```
classifier = OneVsOneClassifier(SVC(kernel='sigmoid'))
```

Результат виконання програми (класифікатор SVM з сигмоїдальним ядром):

```
Accuracy: 57.26%
Precision: 57.1%
Recall: 57.26%
F1: 57.18%
F1 score: 57.18%
<=50K
```

Рис. 2.3. Результат виконання програми (класифікатор SVM з сигмоїдальним ядром)

Висновок, який з видів SVM найкраще виконує завдання класифікації за результатами тренування

Як видно, найкращі результати, приблизно однакові, показують класифікатор SVM з поліноміальним ядром та класифікатор SVM з гаусовим ядром. Однак останній має велику перевагу в ефективності. Класифікатор SVM з сигмоїдальним ядром, за результатами видно, відстає від перших двох за всіма метриками. Треба враховувати, що сигмоїдальне ядро застосовується в ситуаціях, коли структура набору даних не є чіткою, і взаємозв'язки можуть бути складними.

Отже, для нашого випадку, найкраще підходять класифікатори **SVM з поліноміальним ядром та гаусовим ядром**.

Завдання 2.3. **Порівняння якості класифікаторів на прикладі класифікації сортів ірисів**

Необхідно класифікувати сорти ірисів за деякими їх характеристиками: довжина та ширина пелюсток, а також довжина та ширина чашолистків.

Також, в наявності є вимірювання цих же характеристик ірисів, які раніше дозволили досвідченому експерту віднести їх до сортів: setosa, versicolor і virginica.

Використовувати класичний набір даних у машинному навчанні та статистиці - Iris. Він включений у модуль datasets бібліотеки scikit-learn.

Виведення значень ознак для перших п’яти прикладів:

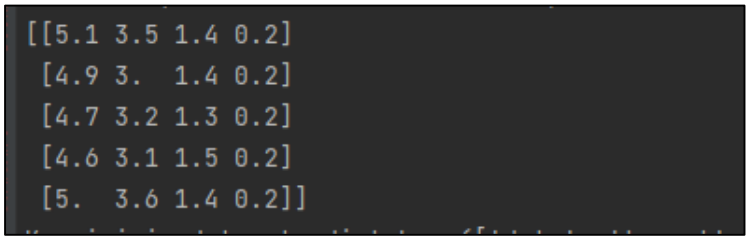

Рис. 3.1. Виведення значень ознак для перших п’яти прикладів

Лістинг коду для ознайомлення зі структурою даних:

```
from sklearn.datasets import load_iris

iris_dataset = load_iris()

print(iris_dataset['data'][:5])

print(f'Ключі iris_dataset: {iris_dataset.keys()}')
print(iris_dataset['DESCR'][:193] + "\n....")
print(f"Назви відповідей: {iris_dataset['target_names']}")
print(f"Назва ознак: {iris_dataset['feature_names']}")
print(f"Тип масиву data: {type(iris_dataset['data'])}")
print(f"Форма масиву data: {iris_dataset['data'].shape}")
print("Відповіді:\n{}".format(iris_dataset['target']))
```

Результат виконання програми:

```
E:\programs\python\python.exe "E:\AI-subject\L-2 Comparison\LR_2_task_3.py"
Ключі iris_dataset: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
.. _iris_dataset:

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

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, pre
....
Назви відповідей: ['setosa' 'versicolor' 'virginica']
Назва ознак: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Тип масиву data: <class 'numpy.ndarray'>
Форма масиву data: (150, 4)
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 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]
```

Рис. 3.2. Результат виконання програми

Лістнг програми після додавання коду для візуалізації даних:

```
from sklearn.datasets import load_iris
import numpy as np
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
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.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

iris_dataset = load_iris()
print(f'Ключі iris_dataset: {iris_dataset.keys()}')
print(iris_dataset['DESCR'][:193] + "\n....")
print(f"Назви відповідей: {iris_dataset['target_names']}")
print(f"Назва ознак: {iris_dataset['feature_names']}")
print(f"Тип масиву data: {type(iris_dataset['data'])}")
print(f"Форма масиву data: {iris_dataset['data'].shape}")
print("Відповіді:\n{}".format(iris_dataset['target']))

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

# shape print(dataset.shape)

print(dataset.head(20))

print(dataset.describe())

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

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

dataset.hist()
pyplot.show()

scatter_matrix(dataset)
pyplot.show()
```

Результат виконання програми:

```
    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
13           4.3          3.0           1.1          0.1  Iris-setosa
14           5.8          4.0           1.2          0.2  Iris-setosa
15           5.7          4.4           1.5          0.4  Iris-setosa
16           5.4          3.9           1.3          0.4  Iris-setosa
17           5.1          3.5           1.4          0.3  Iris-setosa
18           5.7          3.8           1.7          0.3  Iris-setosa
19           5.1          3.8           1.5          0.3  Iris-setosa
       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
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

Рис. 3.3. Результат виконання програми

Діаграма розмаху:

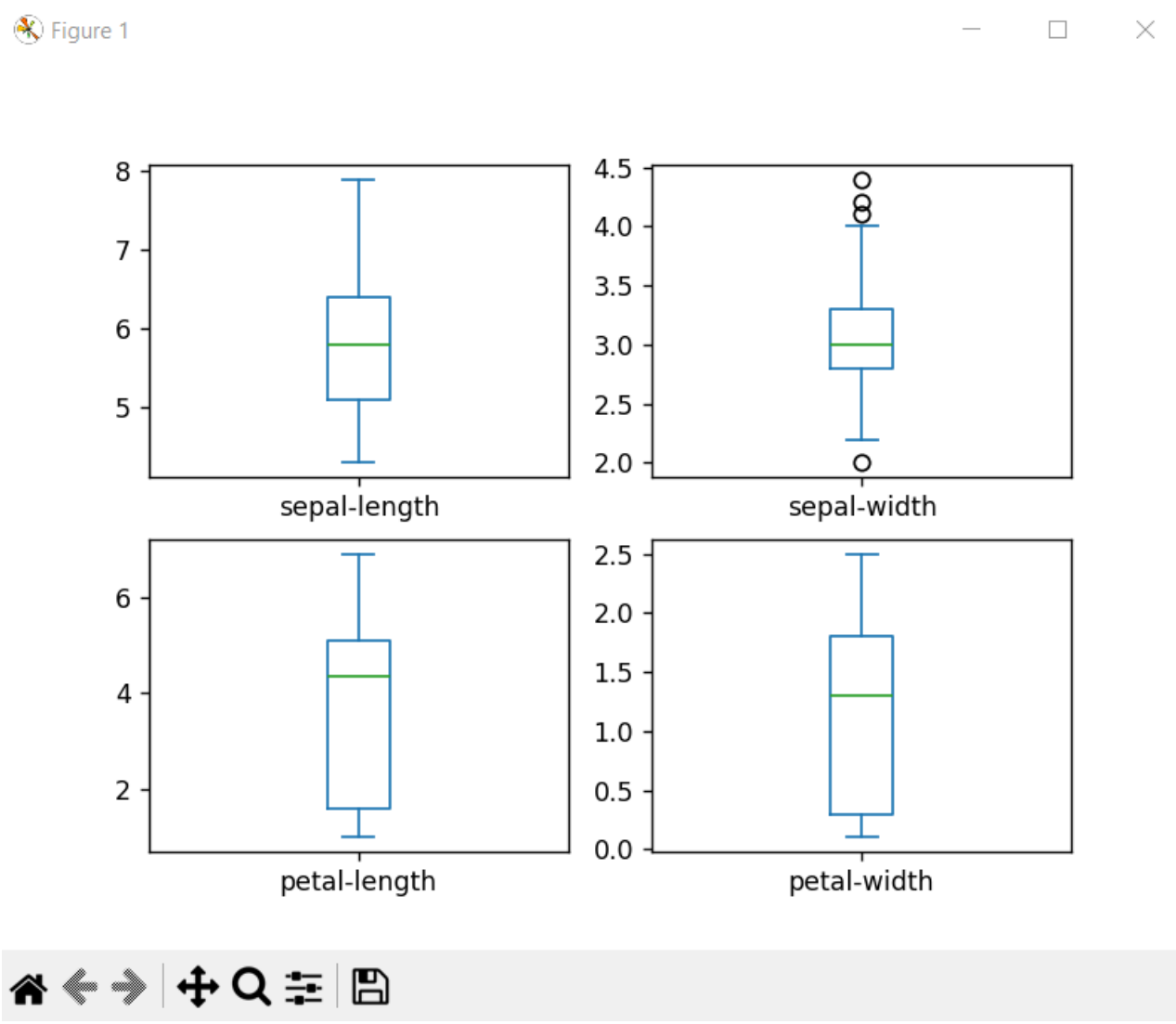

Рис. 3.4. Діаграма розмаху

Гістограма розподілу атрибутів датасета:

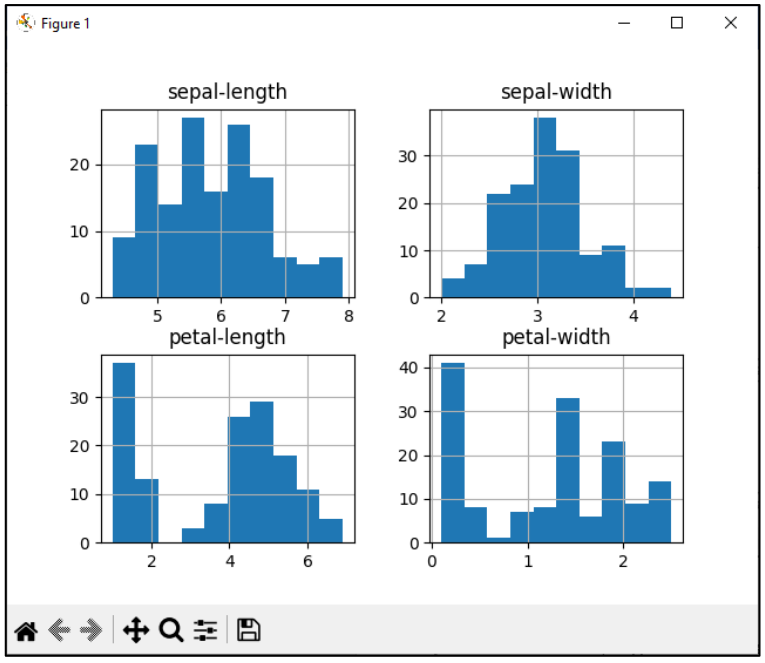

Рис. 3.5. Гістограма розподілу атрибутів датасета

Матриця діаграм розсіювання:

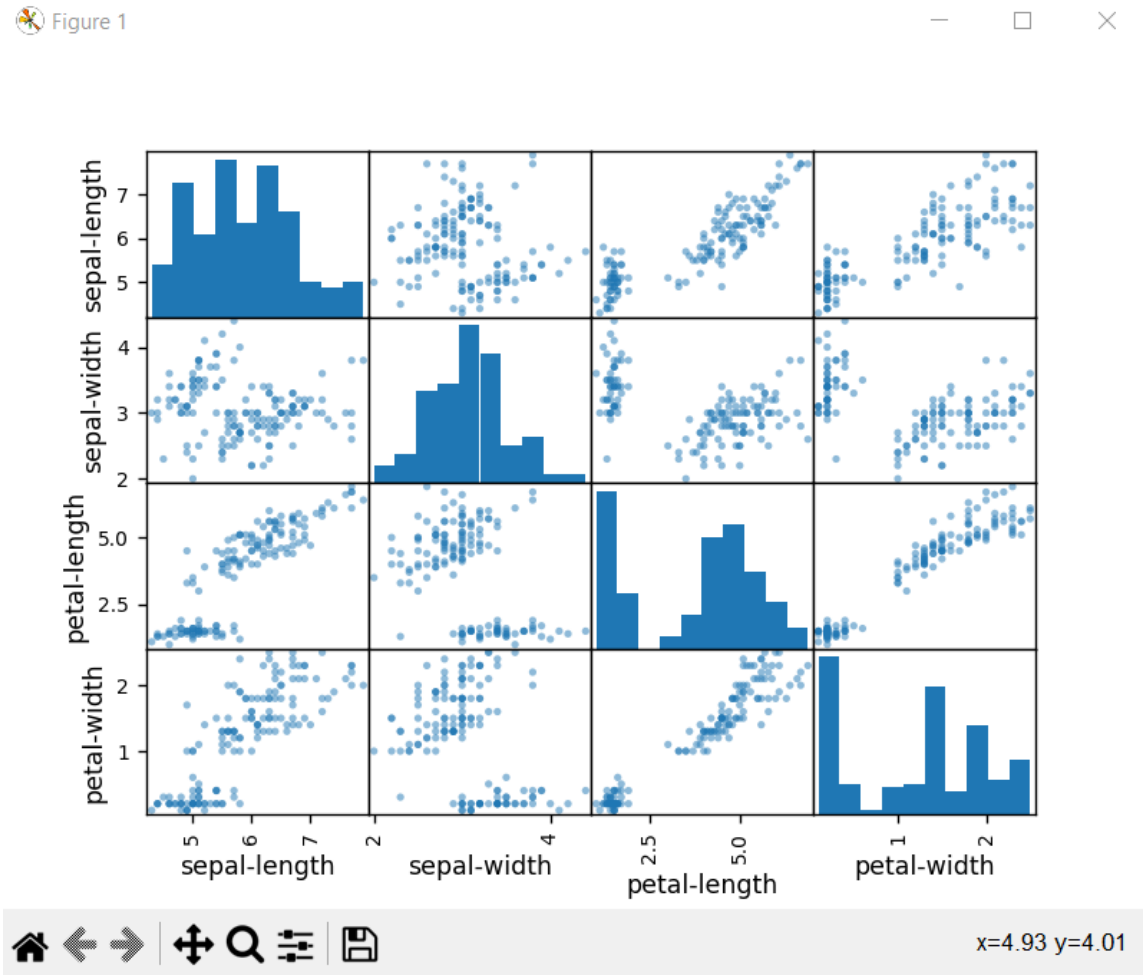

Рис. 3.6. Матриця діаграм розсіювання

Лістингу програми після додавання тестового та навчального наборів та порівняння шести різних алгоритмів:

```
from sklearn.datasets import load_iris
import numpy as np
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
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.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

iris_dataset = load_iris()
print(f'Ключі iris_dataset: {iris_dataset.keys()}')
print(iris_dataset['DESCR'][:193] + "\n....")
print(f"Назви відповідей: {iris_dataset['target_names']}")
print(f"Назва ознак: {iris_dataset['feature_names']}")
print(f"Тип масиву data: {type(iris_dataset['data'])}")
print(f"Форма масиву data: {iris_dataset['data'].shape}")
print("Відповіді:\n{}".format(iris_dataset['target']))

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

# shape print(dataset.shape)

print(dataset.head(20))

print(dataset.describe())

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

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

dataset.hist()
pyplot.show()

scatter_matrix(dataset)
pyplot.show()

array = dataset.values

X = array[:, 0:4]

y = array[:, 4]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
test_size=0.20, random_state=1)

models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
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, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scor-
ing='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()
```

Результат виконання програми:

```
LR: 0.941667 (0.065085)
LDA: 0.975000 (0.038188)
KNN: 0.958333 (0.041667)
CART: 0.933333 (0.050000)
NB: 0.950000 (0.055277)
SVM: 0.983333 (0.033333)
```

Рис. 3.7. Результат виконання програми

Порівняння шести різних алгоритмів:

Логістична регресія або логіт-модель (LR)

Лінійний дискримінантний аналіз (LDA)

Метод k-найближчих сусідів (KNN)

Класифікація та регресія за допомогою дерев (CART)

Наївний баєсовський класифікатор (NB)

Метод опорних векторів (SVM)

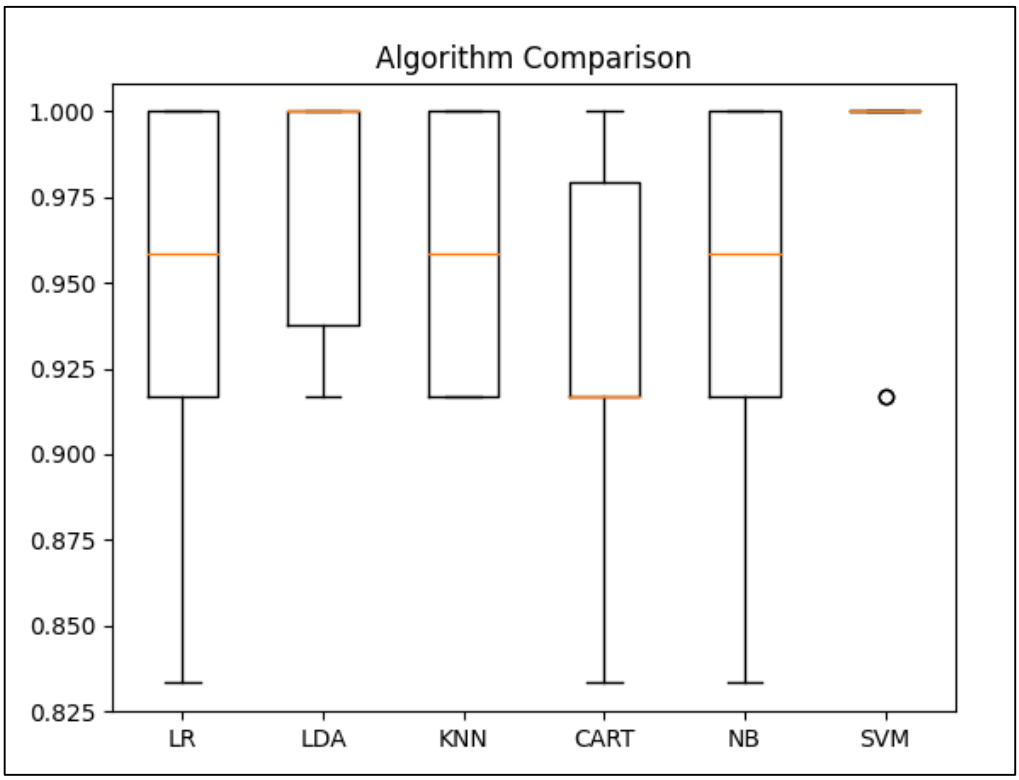

Рис. 3.8. Порівняння шести різних алгоритмів

Вибір та обґрунтування чому обраний метод класифікації є найкращим

Найкращою моделлю для цього набору даних є Метод опорних векторів (SVM), оскільки він має найвищий середній показник точності (0.983333) та найнижче стандартне відхилення (0.033333) серед усіх моделей. Лінійний дискримінантний аналіз (LDA) також продемонстрував високий рівень точності (0.975000) та низьке стандартне відхилення (0.038188), що робить його дуже ефективним.

Отже, для цього набору даних ефективно використовувати **Метод опорних векторів (SVM)** та **Лінійний дискримінантний аналіз (LDA)**

Лістинг програми після додавання коду для отримання прогнозу:

```
from sklearn.datasets import load_iris
import numpy as np
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
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.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

iris_dataset = load_iris()
print(f'Ключі iris_dataset: {iris_dataset.keys()}')
print(iris_dataset['DESCR'][:193] + "\n....")
print(f"Назви відповідей: {iris_dataset['target_names']}")
print(f"Назва ознак: {iris_dataset['feature_names']}")
print(f"Тип масиву data: {type(iris_dataset['data'])}")
print(f"Форма масиву data: {iris_dataset['data'].shape}")
print("Відповіді:\n{}".format(iris_dataset['target']))

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

# shape print(dataset.shape)

print(dataset.head(20))

print(dataset.describe())
```

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

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

dataset.hist()
pyplot.show()

scatter_matrix(dataset)
pyplot.show()

array = dataset.values

X = array[:, 0:4]

y = array[:, 4]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
test_size=0.20, random_state=1)

models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
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, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scor-
ing='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()

model = SVC(gamma='auto')
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))

X_new = np.array([[5, 2.9, 1, 0.2]])

print("Форма масиву X_new: {}".format(X_new.shape))
prediction = model.predict(X_new)
print("Прогноз: {}".format(prediction))
print("Спрогнозована мітка: {}".format(prediction[0]))
```

Результат виконання програми:

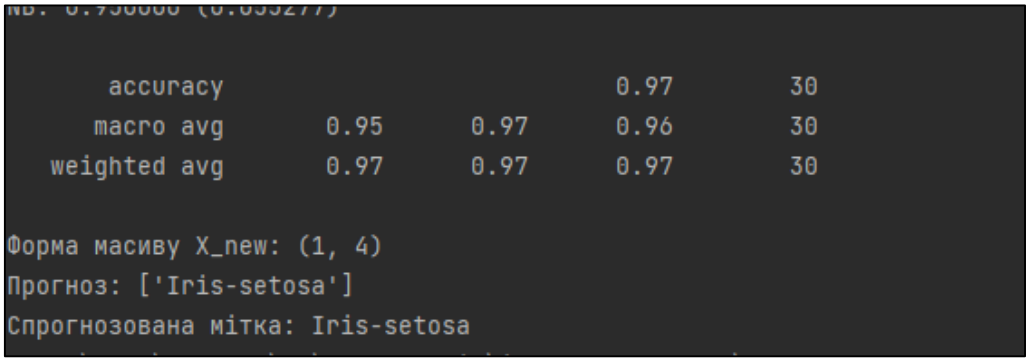

Рис. 3.9. Результат виконання програми

Висновки щодо того яку якість класифікації за результатами тренування вдалося досягти та до якого класу належить квітка

Квітка належить до класу **Iris-Setosa**. Вдалося досягти **0.97** показника якості.

Завдання 2.4. **Порівняння якості класифікаторів для набору даних завдання 2.1**

По аналогії із завданням 2.3 створіть код для порівняння якості класифікації набору даних income_data.txt (із завдання 2.1) різними алгоритмами. Використати такі алгоритми класифікації:

Логістична регресія або логіт-модель (LR)

Лінійний дискримінантний аналіз (LDA)

Метод k-найближчих сусідів (KNN)

Класифікація та регресія за допомогою дерев (CART)

Наївний баєсовський класифікатор (NB)

Метод опорних векторів (SVM)

Розрахуйте показники якості класифікації для кожного алгоритму Порівняйте їх між собою. Оберіть найкращий для рішення задачі. Поясніть чому ви так вирішили у висновках до завдання.

Лістинг програми:

```
import numpy as np
from matplotlib import pyplot
from sklearn import preprocessing
from sklearn.svm import SVC
```

```
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.model_selection import train_test_split, cross_val_score, Strati-
fiedKFold

input_file = 'income_data.txt'

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

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(', ')

        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

X = np.array(X)

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)

scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
X = scaler.fit_transform(X)

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

for name, model in models:
    print(f"Results for {name}:")

    model.fit(X=X, y=y)
```

```
    f1 = cross_val_score(model, X, y, scoring='f1_weighted', cv=3)
    accuracy_values = cross_val_score(model, X, y, scoring='accuracy', cv=3)
    print(f"Accuracy: {round(100 * accuracy_values.mean(), 2)}%")
    precision_values = cross_val_score(model, X, y, scoring='precision_weighted',
cv=3)
    print(f"Precision: {round(100 * precision_values.mean(), 2)}%")
    recall_values = cross_val_score(model, X, y, scoring='recall_weighted', cv=3)
    print(f"Recall: {round(100 * recall_values.mean(), 2)}%")
    f1_values = cross_val_score(model, X, y, scoring='f1_weighted', cv=3)
    print(f"F1: {round(100 * f1_values.mean(), 2)}%")
    print(f"F1 Score: {round(100 * f1.mean(), 2)}%")

    input_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married',
'Handlers-cleaners', 'Not-in-family',
                  'White', 'Male', '0', '0', '40', 'United-States']

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

    predicted class = model.predict(input data encoded)
    print(f"{label_encoder[-1].inverse_transform(predicted_class)[0]}\n")


X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
test_size=0.20, random_state=1)

results = []
names = []

for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scor-
ing='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()
```

Результат виконання програми:

```
Results for LR:
Accuracy: 81.82%
Precision: 80.69%
Recall: 81.82%
F1: 80.25%
F1 Score: 80.25%
```

Рис. 4.1. Результати класифікатора LR

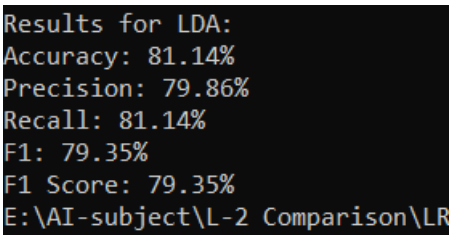

Рис. 4.2. Результати класифікатора LDA

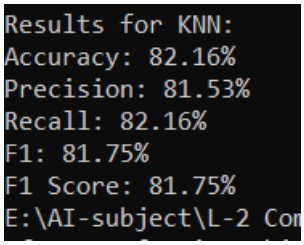

Рис. 4.3. Результати класифікатора KNN

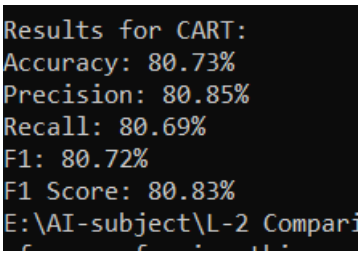

Рис. 4.4. Результати класифікатора CART

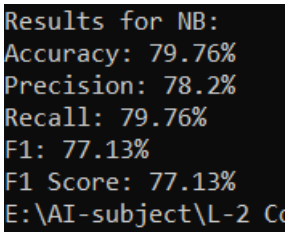

Рис. 4.5. Результати класифікатора NB

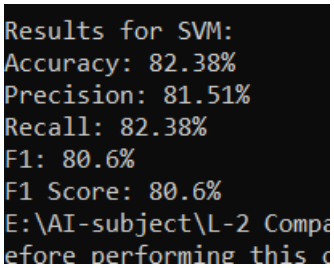

Рис. 4.6. Результати класифікатора SVM

```
  input_data_encoded[i] =
>50K

LR: 0.818849 (0.004427)
LDA: 0.812176 (0.003802)
KNN: 0.817606 (0.003760)
CART: 0.805586 (0.007468)
NB: 0.799080 (0.005377)
SVM: 0.824112 (0.005380)
```

Рис. 4.7. Результат вконання програми (порівняння алгоритмів)

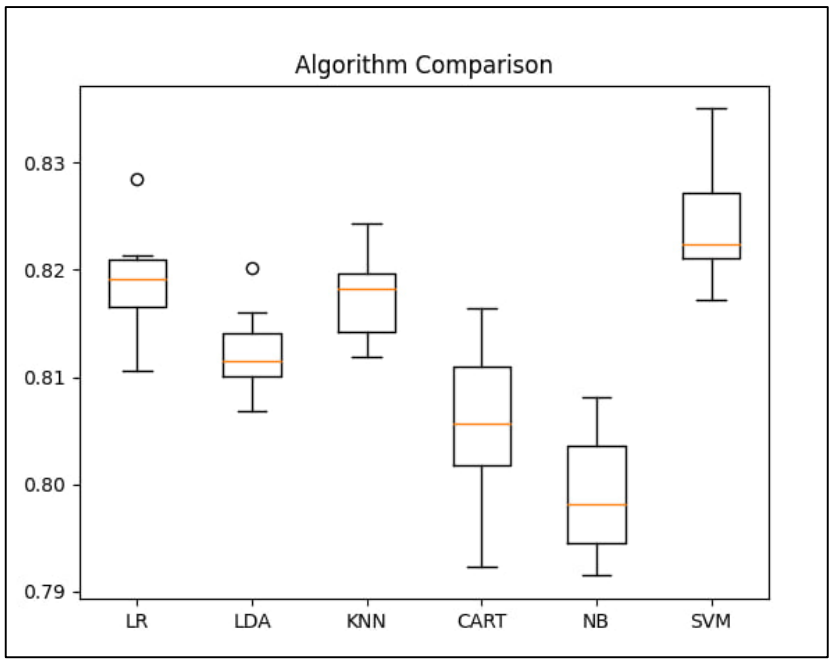

Рис. 4.8. Графік порівняння алгоритмів

Результати порівняння досліджуваних алгоритмів

1. **SVM (Метод опорних векторів)** має найвищий середній показник точності - 82.41%, що свідчить про його потенційну ефективність у класифікації даних.
2. **K-Nearest Neighbors (KNN)** та **LR (Логістична регресія)** також демонструють високі показники точності, відповідно 81.76% та 81.88%.

3. **LDA (Дискримінантний аналіз)** показує добрі результати, але трохи нижчі, з точністю 81.22%.
4. **CART (Класифікаційне дерево)** та **NB (Наївний Баєсівський класифікатор)** показують меншу точність, відповідно 80.55% та 79.91%.

З урахуванням усіх цих даних, найефективнішим алгоритмом для цього конкретного набору даних може бути **SVM**, оскільки він має найвищу середню точність. Проте, враховуючи, те, що він виконується набагато повільніше, краще скористатися **K-Nearest Neighbors (KNN)** або **Logistic Regression (LR),** які демонструють подібно високу точність, але можуть працювати швидше.

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

Виправте код та виконайте класифікацію. Опишіть які налаштування класифікатора Ridge тут використані та що вони позначають. Опишіть які показники якості використовуються та їх отримані результати. Вставте у звіт та поясніть зображення Confusion.jpg. Опишіть, що таке коефіцієнт Коена Каппа та коефіцієнт кореляції Метьюза. Що вони тут розраховують та що показують.

Лістинг програми:

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

sns.set()
iris = load_iris()
X, y = iris.data, iris.target
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.3, ran-
dom_state=0)

clf = RidgeClassifier(tol=1e-2, solver="sag")

clf.fit(Xtrain, ytrain)
ypred = clf.predict(Xtest)

print('Accuracy:', np.round(metrics.accuracy_score(ytest, ypred), 4))
print('Precision:', np.round(metrics.precision_score(ytest, ypred, aver-
age='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))
print('\t\tClassification Report:\n', metrics.classification_report(ypred, ytest))

mat = confusion_matrix(ytest, ypred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.savefig("Confusion.jpg")

f = BytesIO()
plt.savefig(f, format="svg")
```

Результат виконання програми:

```
E:\AI-subject\L-2 Comparison>python LR_2_task_5.py
Accuracy: 0.7556
Precision: 0.8333
Recall: 0.7556
F1 Score: 0.7503
Cohen Kappa Score: 0.6431
Matthews Corrcoef: 0.6831
                Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.44      0.89      0.59         9
           2       0.91      0.50      0.65        20

    accuracy                           0.76        45
   macro avg       0.78      0.80      0.75        45
weighted avg       0.85      0.76      0.76        45


E:\AI-subject\L-2 Comparison>
```

Рис. 5.1. Результат виконання програми

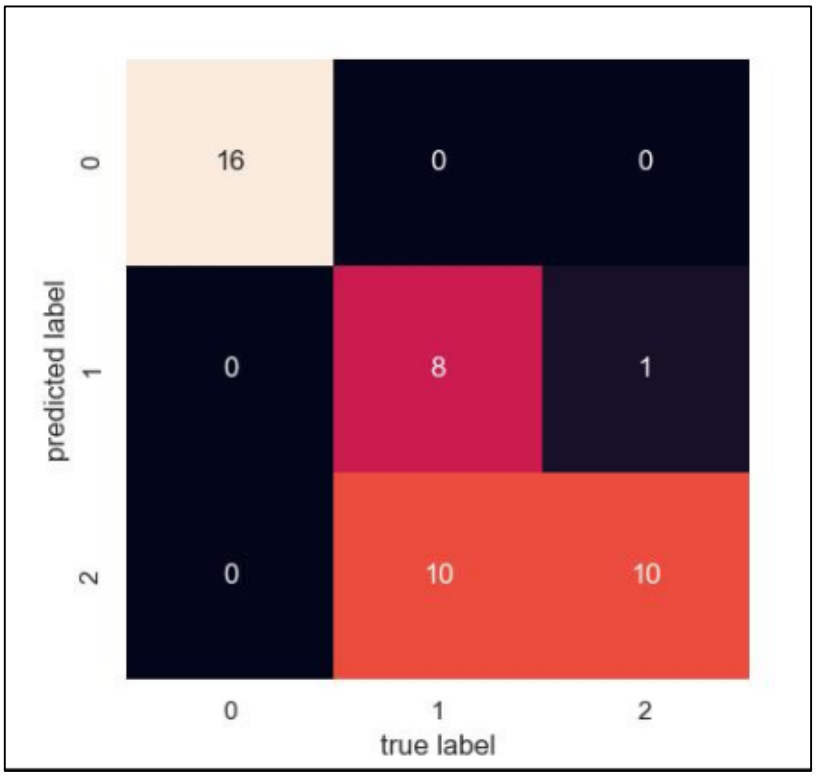

Рис. 5.2. Зображення Confusion.jpg

Пояснення та опис виконання завдання

Основні налаштування:

1. **tol=1e-2**: Цей параметр визначає критерій зупинки для оптимізаційного алгоритму. У цьому випадку, оптимізаційний алгоритм (SAG - Stochastic Average Gradient Descent) припиняє роботу, коли зміна ваг ваги стає меншою за **tol**.
2. **solver="sag"**: Вказує, який оптимізаційний алгоритм використовувати для навчання класифікатора. У цьому випадку, обраний SAG.

Показники якості:

1. **Accuracy**
2. **Precision**
3. **Recall**
4. **F1 Score**

5. **Cohen Kappa Score**: Це міра узгодженості між прогнозованими та спостережуваними класами, враховуючи ймовірність випадкового узгодження.
6. **Matthews Correlation Coefficient**: Це міра узгодженості, особливо враховуючи незбалансованість класів.

Матриця плутанини – це таблиця особливого компонування, що дає можливість унаочнювати продуктивність алгоритму, зазвичай керованого навчання. Кожен з рядків цієї матриці представляє зразки прогнозованого класу, тоді як кожен зі стовпців представляє зразки справжнього класу (або навпаки). Її назва походить від того факту, що вона дає можливість просто бачити, чи допускає система невідповідності між цими двома класами.

Коефіцієнт Коена Каппа та коефіцієнт кореляції Метьюза використовуються для вимірювання узгодженості між прогнозованими та спостережуваними класами, при цьому вони враховують ймовірність випадкового узгодження.

Значення цих коефіцієнтів може бути від -1 до 1. Більше значення вказує на кращу узгодженість. Значення нуль вказує на те, що узгодженість зводиться до випадкового узгодження, а від'ємне значення вказує на гіршу узгодженість, ніж випадкове узгодження.

Висновок: у ході виконання лабораторної роботи я, використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити різні методи класифікації даних та навчився їх порівнювати.