ЛАБОРАТОРНА РОБОТА№ 2

ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ

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

Завдання 1:

Лістинг коду:

```
import numpy as np
import matplotlib.pyplot as plt
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
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.preprocessing import StandardScaler

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

# Читання даних
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.strip().split(', ')

       if data[-1] == '<=50K' and count_class1 < max_datapoints:
```

```
            X.append(data[:-1])  # Додамо всі елементи, крім останнього, до
X
            y.append(0)  # Мітка класу <=50K
            count_class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data[:-1])
            y.append(1)  # Мітка класу >50K
            count_class2 += 1

# Перетворення на масив numpy
X = np.array(X)
y = np.array(y)

# Перевірка на наявність даних
if X.size == 0:
    raise ValueError("Помилка: Дані не були зчитані коректно. Перевірте
формат файлу або умови зчитування.")

# Перетворення рядкових даних на числові
label_encoder = []
X_encoded = np.zeros((X.shape[0], X.shape[1]), dtype=object)  # Задаємо
правильну форму

for i in range(X.shape[1]):
    if np.issubdtype(X[:, i].dtype, np.number):  # Перевірка на числові дані
        X_encoded[:, i] = X[:, i]
    else:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label_encoder.append(le)

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

# Нормалізація даних
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Створення SVM-класифікатора
classifier = OneVsOneClassifier(LinearSVC(random_state=0, max_iter=5000))
```

```
# Розділення даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)

# Навчання класифікатора
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)

# Обчислення показників якості
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',
zero_division=0)
f1 = f1_score(y_test, y_test_pred, average='weighted', zero_division=0)

# Виведення результатів
print(f"Акуратність: {round(100 * accuracy, 2)}%")
print(f"Точність: {round(100 * precision, 2)}%")
print(f"Повнота: {round(100 * recall, 2)}%")


# Обчислення F-міри для SVM-класифікатора
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=2)

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

# Передбачення результату для тестової точки даних
input_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married',
              'Handlers-cleaners', 'Not-in-family', 'White', 'Male', '0',
'0', '40', 'United-States']

# Кодування тестової точки даних
input_data_encoded = [-1] * len(input_data)
count = 0

for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = int(input_data[i])
    else:
        # Спроба кодування значення
```

```
        if count < len(label_encoder):  # Перевірка, чи існує кодер
            try:
                input_data_encoded[i] =
int(label_encoder[count].transform([input_data[i]])[0])
            except ValueError:
                print(f"Попередження: '{item}' не було у навчальних даних,
використано умовне значення -1.")
                input_data_encoded[i] = -1  # Або використовувати інше
стандартне значення
        else:
            print(f"Попередження: '{item}' не було у навчальних даних,
використано умовне значення -1.")
            input_data_encoded[i] = -1  # Або використовувати інше
стандартне значення
    count += 1

# Обрізаємо до 13 ознак, якщо потрібно
input_data_encoded = input_data_encoded[:13]  # Обрізаємо до 13 ознак

# Використання класифікатора для кодованої точки даних та виведення
результату
predicted_class = classifier.predict([input_data_encoded])  # Додаємо
квадратні дужки для 2D форми
print(label_encoder[-1].inverse_transform(predicted_class)[0])
```

Результат виконання програми:

```
Акуратність: 90.78%
Точність: 82.42%
Повнота: 90.78%
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklea
lated class in y has only 1 members, which is less than n_splits=2.
  warnings.warn(
F1 score: 86.98%
Japan
```

Висновок:

- **Акуратність**: 90.78%
- **Точність**: 82.42%
- **Повнота**: 90.78%
- **F1 score**: 86.98%

Клас, до якого належить тестова точка, був визначений як **"Japan"**

Завдання №2

Лістинг LR_2_task_2_1.py:

```
import time

import numpy as np

from sklearn import preprocessing

from sklearn.svm import SVC

from sklearn.model_selection import train_test_split

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

from sklearn.preprocessing import StandardScaler




# Вхідний файл, який містить дані

input_file = 'income_data.txt'



# Читання даних

X = []

y = []

count_class1 = 0

count_class2 = 0

max_datapoints = 25000



with open(input_file, 'r') as f:

   for line in f.readlines():
```

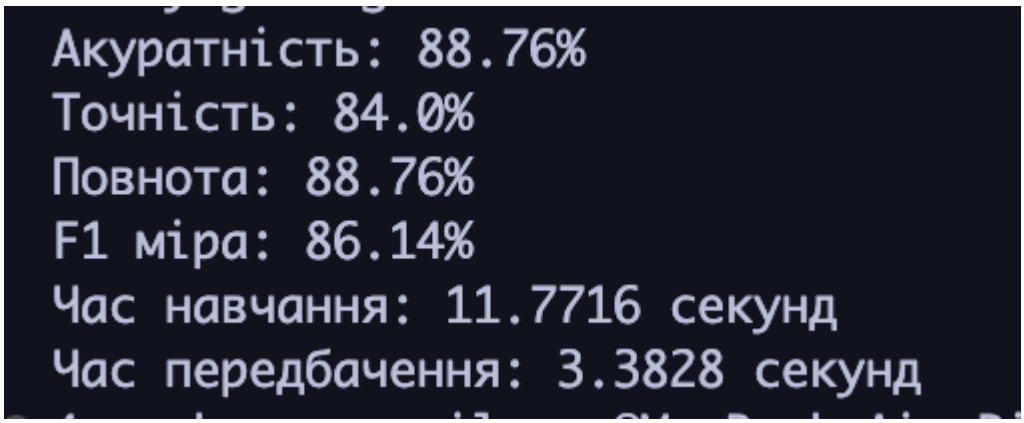
```
        if count_class1 >= max_datapoints and count_class2 >=
max_datapoints:
            break
        if '?' in line:
            continue

        data = line.strip().split(', ')

        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data[:-1])  # Додамо всі елементи, крім останнього, до
X
            y.append(0)  # Мітка класу <=50K
            count_class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data[:-1])
            y.append(1)  # Мітка класу >50K
            count_class2 += 1

# Перетворення на масив numpy
X = np.array(X)
y = np.array(y)

# Перевірка на наявність даних
if X.size == 0:
```

```
    raise ValueError("Помилка: Дані не були зчитані коректно. Перевірте формат файлу або умови зчитування.")



# Перетворення рядкових даних на числові

label_encoder = []

X_encoded = np.zeros((X.shape[0], X.shape[1]), dtype=object)  # Задаємо правильну форму



for i in range(X.shape[1]):

    if np.issubdtype(X[:, i].dtype, np.number):  # Перевірка на числові дані

        X_encoded[:, i] = X[:, i]

    else:

        le = preprocessing.LabelEncoder()

        X_encoded[:, i] = le.fit_transform(X[:, i])

        label_encoder.append(le)



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

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



# Нормалізація даних

scaler = StandardScaler()

X = scaler.fit_transform(X)



# Таймер для оцінки швидкості

start_time = time.time()
```

```python
# Розділення даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)

# Поліноміальне ядро з degree=8
classifier = SVC(kernel='poly', degree=8, random_state=0)
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)

# Оцінка часу навчання
train_time = time.time() - start_time

# Таймер для передбачення
start_time = time.time()
y_test_pred = classifier.predict(X_test)
predict_time = time.time() - start_time

# Обчислення показників якості
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',
zero_division=0)
f1 = f1_score(y_test, y_test_pred, average='weighted', zero_division=0)
```

```
print(f"Акуратність: {round(100 * accuracy, 2)}%")

print(f"Точність: {round(100 * precision, 2)}%")

print(f"Повнота: {round(100 * recall, 2)}%")

print(f"F1 міра: {round(100 * f1, 2)}%")

print(f"Час навчання: {train_time:.4f} секунд")

print(f"Час передбачення: {predict_time:.4f} секунд")
```

Результат виконання завдання:



Лістинг LR_2_task_2_2.py:

```
import time

import numpy as np

from sklearn import preprocessing

from sklearn.svm import SVC

from sklearn.model_selection import train_test_split

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

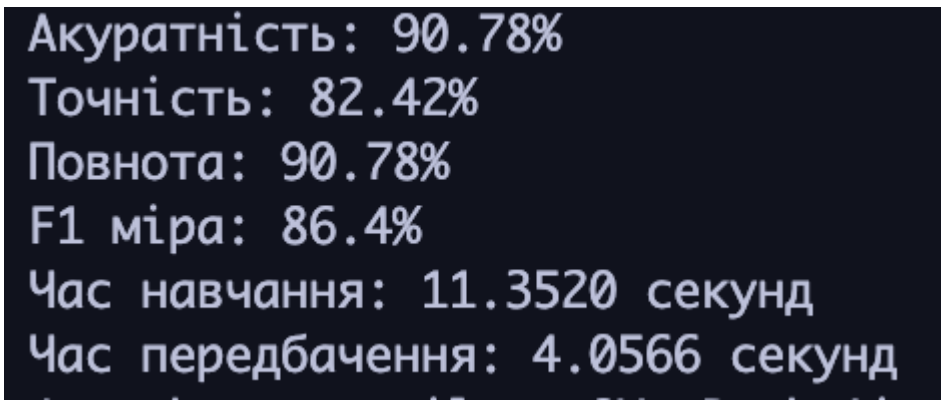
```
from sklearn.preprocessing import StandardScaler



# Вхідний файл, який містить дані

input_file = 'income_data.txt'



# Читання даних

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.strip().split(', ')



       if data[-1] == '<=50K' and count_class1 < max_datapoints:

           X.append(data[:-1])  # Додамо всі елементи, крім останнього, до
X
```

```
            y.append(0)  # Мітка класу <=50K
            count_class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data[:-1])
            y.append(1)  # Мітка класу >50K
            count_class2 += 1

# Перетворення на масив numpy
X = np.array(X)
y = np.array(y)

# Перевірка на наявність даних
if X.size == 0:
    raise ValueError("Помилка: Дані не були зчитані коректно. Перевірте формат файлу або умови зчитування.")

# Перетворення рядкових даних на числові
label_encoder = []
X_encoded = np.zeros((X.shape[0], X.shape[1]), dtype=object)  # Задаємо правильну форму

for i in range(X.shape[1]):
    if np.issubdtype(X[:, i].dtype, np.number):  # Перевірка на числові дані
        X_encoded[:, i] = X[:, i]
    else:
```

```
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label_encoder.append(le)

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

# Нормалізація даних
scaler = StandardScaler()
X = scaler.fit_transform(X)
# Таймер для оцінки швидкості
start_time = time.time()

# Розділення даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)

# Гаусове ядро
classifier = SVC(kernel='rbf', random_state=0)
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)

# Оцінка часу навчання
train_time = time.time() - start_time
```

```python
# Таймер для передбачення

start_time = time.time()

y_test_pred = classifier.predict(X_test)

predict_time = time.time() - start_time



# Обчислення показників якості

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',
zero_division=0)

f1 = f1_score(y_test, y_test_pred, average='weighted', zero_division=0)



print(f"Акуратність: {round(100 * accuracy, 2)}%")

print(f"Точність: {round(100 * precision, 2)}%")

print(f"Повнота: {round(100 * recall, 2)}%")

print(f"F1 міра: {round(100 * f1, 2)}%")
```

Результат виконання завдання:



Лістинг LR_2_task_2_3.py:

```
import time

import numpy as np

from sklearn import preprocessing

from sklearn.svm import SVC

from sklearn.model_selection import train_test_split

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

from sklearn.preprocessing import StandardScaler



# Вхідний файл, який містить дані

input_file = 'income_data.txt'



# Читання даних

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:
```

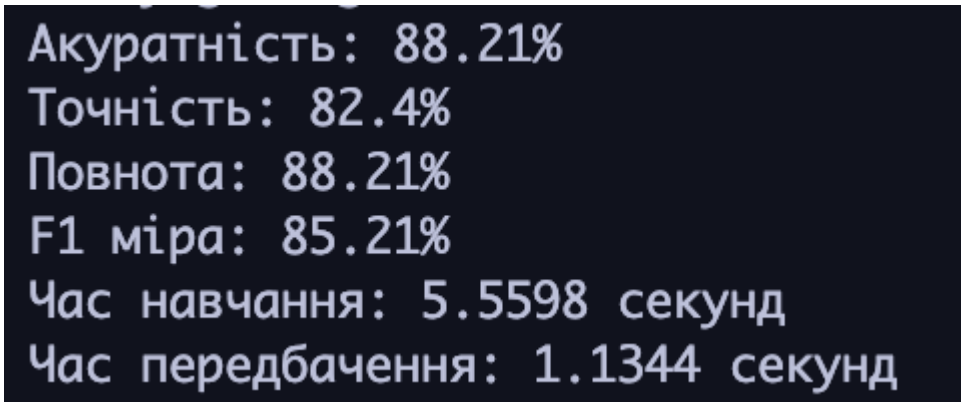
```
            break
        if '?' in line:
            continue

        data = line.strip().split(', ')

        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data[:-1])  # Додамо всі елементи, крім останнього, до
X
            y.append(0)  # Мітка класу <=50K
            count_class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data[:-1])
            y.append(1)  # Мітка класу >50K
            count_class2 += 1

# Перетворення на масив numpy
X = np.array(X)
y = np.array(y)

# Перевірка на наявність даних
if X.size == 0:
   raise ValueError("Помилка: Дані не були зчитані коректно. Перевірте
формат файлу або умови зчитування.")
```

```
# Перетворення рядкових даних на числові

label_encoder = []

X_encoded = np.zeros((X.shape[0], X.shape[1]), dtype=object)  # Задаємо правильну форму



for i in range(X.shape[1]):

   if np.issubdtype(X[:, i].dtype, np.number):  # Перевірка на числові дані

       X_encoded[:, i] = X[:, i]

   else:

       le = preprocessing.LabelEncoder()

       X_encoded[:, i] = le.fit_transform(X[:, i])

       label_encoder.append(le)



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

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



# Нормалізація даних

scaler = StandardScaler()

X = scaler.fit_transform(X)



# Таймер для оцінки швидкості

start_time = time.time()



# Розділення даних на навчальний та тестовий набори
```

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



# Сигмоїдальне ядро

classifier = SVC(kernel='sigmoid', random_state=0)

classifier.fit(X_train, y_train)

y_test_pred = classifier.predict(X_test)



# Оцінка часу навчання

train_time = time.time() - start_time



# Таймер для передбачення

start_time = time.time()

y_test_pred = classifier.predict(X_test)

predict_time = time.time() - start_time



# Обчислення показників якості

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',
zero_division=0)

f1 = f1_score(y_test, y_test_pred, average='weighted', zero_division=0)



print(f"Акуратність: {round(100 * accuracy, 2)}%")
```

```python
print(f"Точність: {round(100 * precision, 2)}%")

print(f"Повнота: {round(100 * recall, 2)}%")

print(f"F1 міра: {round(100 * f1, 2)}%")

print(f"Час навчання: {train_time:.4f} секунд")

print(f"Час передбачення: {predict_time:.4f} секунд")
```

Результат виконання завдання:



Висновки:

Дивлячись на результати які видали програми, то **Гаусове ядро** показало найкращі результати серед усіх типів за показниками якості класифікації. Це свідчить про те, що гаусове ядро краще розділяє класи, особливо в нелінійних випадках, що позитивно позначається на точності та повноті класифікації.

Поліноміальне ядро показало результати близькі до гаусового, але трохи поступається за всіма показниками. Час навчання і час передбачення є трохи більшими, ніж для гаусового ядра. Це може бути корисно для складних задач, але гаусове ядро загалом виявилося ефективнішим.

Сигмоїдальне ядро має найменші значення для всіх показників якості класифікації, але показало значну перевагу в часі навчання і передбачення. Воно може бути хорошим вибором для задач, де важливий швидкий час виконання, але деяке зниження точності є допустимим.

Завдання 3

Ознайомлення зі структурою данних

Лістинг:

```python
from sklearn.datasets import load_iris

iris_dataset = load_iris()

print("Ключі iris_dataset: \n{}".format(iris_dataset.keys()))

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

print("Назви відповідей: \n{}".format(iris_dataset['target_names']))

print("Назва ознак: \n{}".format(iris_dataset['feature_names']))

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

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

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

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

Результат виконання:

```
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, predictive
...
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)
Тип масиву target:
<class 'numpy.ndarray'>
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 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
 2 2]
```

Лістинг:

```
# Завантаження бібліотек

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



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)

print(dataset.shape)

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

```
print(dataset.describe())

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



# Діаграма розмаху

dataset.plot(kind='box', subplots=True, layout=(2,2),

sharex=False, sharey=False)

pyplot.show()



# Гістограма розподілу атрибутів датасета

dataset.hist()

pyplot.show()



#Матриця діаграм розсіювання

scatter_matrix(dataset)

pyplot.show()
```

Результат виконання:

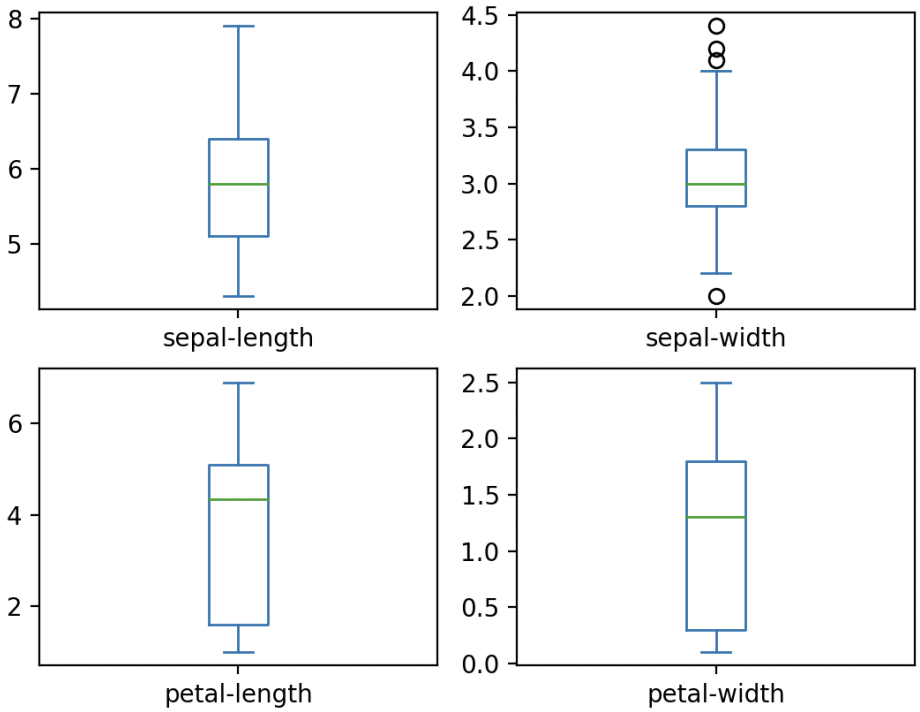


Рис.1. Діаграма розмаху

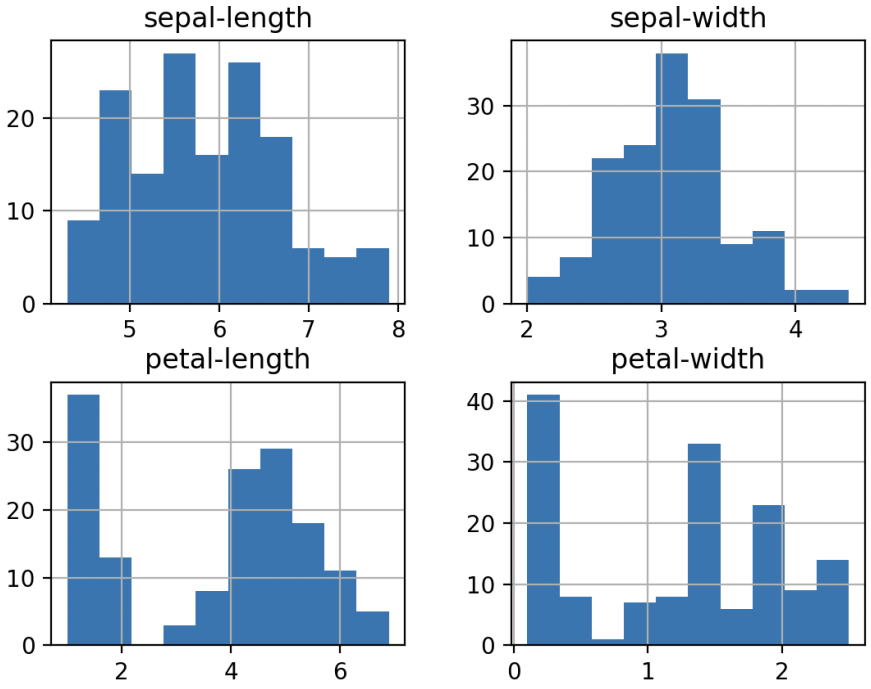


Рис.2. Гістограма розподілу атрибутів датасета

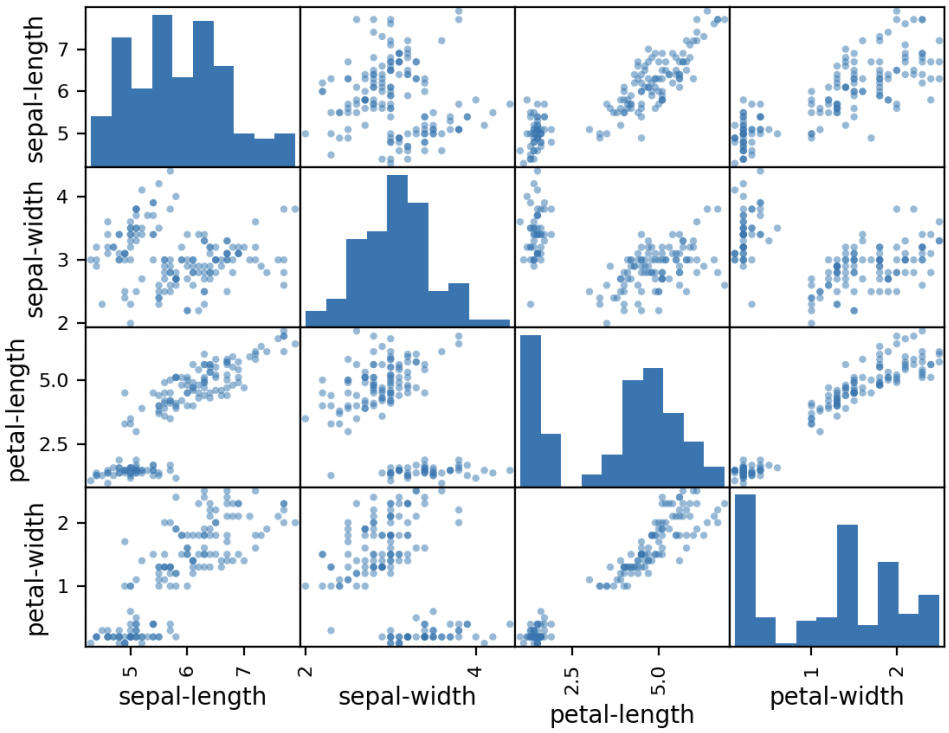


Рис.3. Матриця діаграм розсіювання

Лістинг LR_2_task_3.py:

```
# Завантаження бібліотек

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

from sklearn.multiclass import OneVsRestClassifier



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)

""" print(dataset.shape)

print(dataset.head(20))

print(dataset.describe())

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



""" # Діаграма розмаху

dataset.plot(kind='box', subplots=True, layout=(2,2),

sharex=False, sharey=False)

pyplot.show()



# Гістограма розподілу атрибутів датасета

dataset.hist()
```

```
pyplot.show()



#Матриця діаграм розсіювання

scatter_matrix(dataset)

pyplot.show() """



# Розділення датасету на навчальну та контрольну вибірки

array = dataset.values



# Вибір перших 4-х стовпців

X = array[:,0:4]



# Вибір 5-го стовпця

Y = array[:,4]



# Поділ X та Y на навчальну та контрольну вибірки

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




# Завантажуємо алгоритми моделі

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')))



# оцінюємо модель на кожній ітерації

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,
scoring='accuracy')

   results.append(cv_results)

   names.append(name)

   print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))



# Порівняння алгоритмів

pyplot.boxplot(results, labels=names)

pyplot.title('Algorithm Comparison')

pyplot.show()
```

Результат виконання роботи:

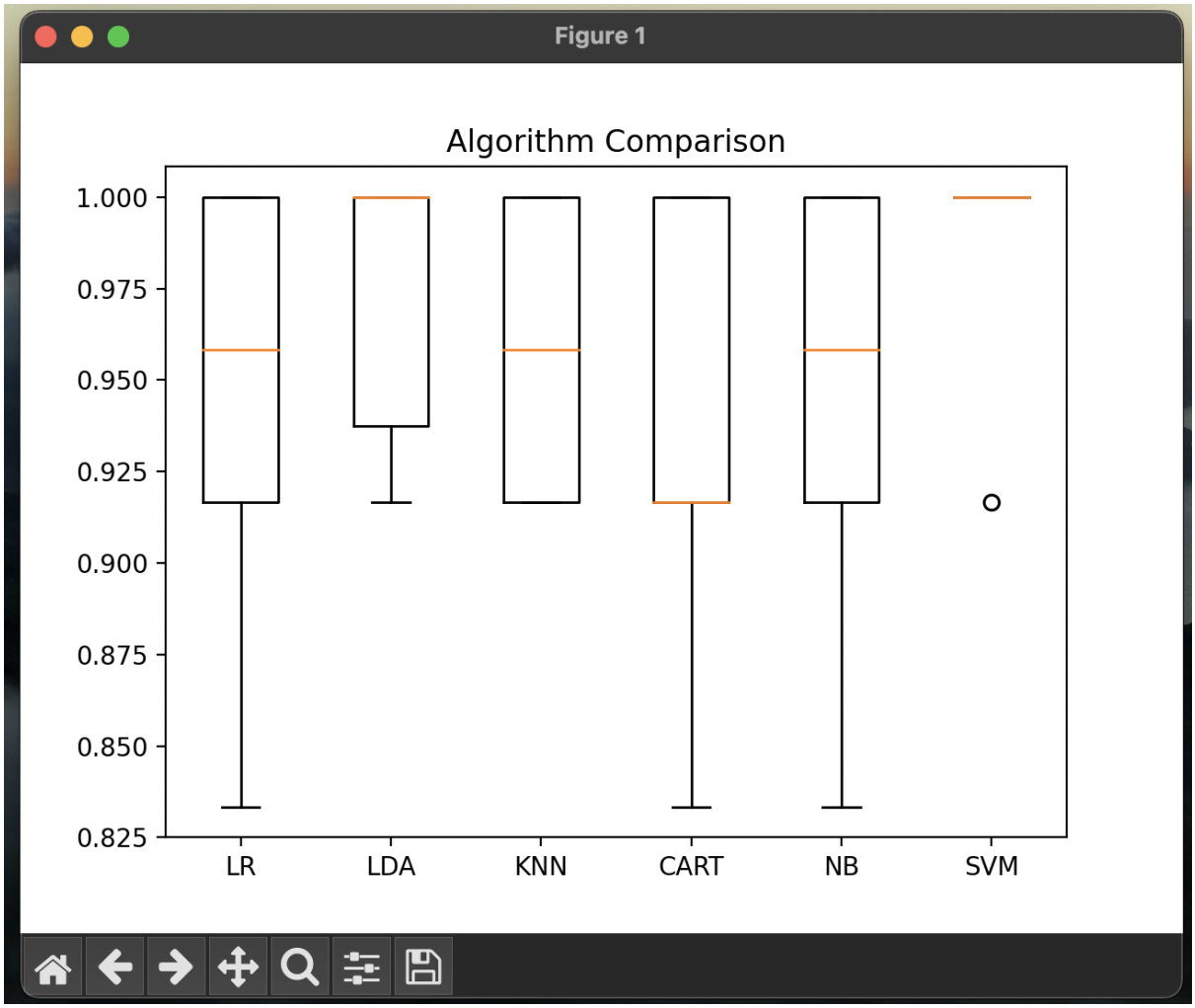


Рис.4. Графік порівняння алгоритмів

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

Рис.5. Отримані результати алгоритмів

Висновки:

Алгоритм	Середня точність (Accuracy)	Стандартне відхилення
LR	94.17%	6.51%
LDA	97.50%	3.82%
KNN	95.83%	4.17%
CART	94.17%	5.34%
NB	95.00%	5.53%
SVM	**98.33%**	**3.33%**

Метод SVM є оптимальним вибором для класифікації датасету Iris. Він перевершує інші алгоритми за точністю та стабільністю, що робить його підходящим для задач, які вимагають високої надійності моделі.

Лістринг:

```
# Завантаження бібліотек

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

from sklearn.multiclass import OneVsRestClassifier

import numpy as np

from sklearn.neighbors import KNeighborsClassifier



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)

""" print(dataset.shape)

print(dataset.head(20))

print(dataset.describe())

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

```
""" # Діаграма розмаху

dataset.plot(kind='box', subplots=True, layout=(2,2),

sharex=False, sharey=False)

pyplot.show()



# Гістограма розподілу атрибутів датасета

dataset.hist()

pyplot.show()



#Матриця діаграм розсіювання

scatter_matrix(dataset)

pyplot.show() """



# Розділення датасету на навчальну та контрольну вибірки

array = dataset.values



# Вибір перших 4-х стовпців

X = array[:,0:4]



# Вибір 5-го стовпця

Y = array[:,4]



# Поділ X та Y на навчальну та контрольну вибірки

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

```
# Завантажуємо алгоритми моделі
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')))

# оцінюємо модель на кожній ітерації
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,
scoring='accuracy')
   results.append(cv_results)
   names.append(name)
   print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

# Порівняння алгоритмів
pyplot.boxplot(results, labels=names)
```

```
pyplot.title('Algorithm Comparison')

pyplot.show()



# Створюємо прогноз на контрольній вибірці

model = SVC(gamma='auto')

model.fit(X_train, Y_train)

predictions = model.predict(X_validation)



# Оцінюємо прогноз

print(accuracy_score(Y_validation, predictions))

print(confusion_matrix(Y_validation, predictions))

print(classification_report(Y_validation, predictions))



# Завантаження і підготовка даних

iris_dataset = {

   'target_names': np.array(['setosa', 'versicolor', 'virginica'])

}



# Навчання моделі KNN

knn = KNeighborsClassifier()

knn.fit(X_train, Y_train)



# Новий екземпляр квітки

X_new = np.array([[5, 2.9, 1, 0.2]])  # Довжина і ширина чашолистка,
довжина і ширина пелюстки
```

```
print("Форма масиву X_new: {}".format(X_new.shape))



# Прогнозування

prediction = knn.predict(X_new)

predicted_class = iris_dataset['target_names'][np.where(np.unique(Y_train)
== prediction[0])[0][0]]



# Виведення результату

print("Прогноз: {}".format(prediction))

print("Спрогнозована мітка: {}".format(predicted_class))
```

Результат виконання:

```
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

Форма масиву X_new: (1, 4)
Прогноз: ['Iris-setosa']
Спрогнозована мітка: setosa
```

1. **Звіт класифікації:**
   - *Iris-setosa*: 100% точність, 100% повнота
   - *Iris-versicolor*: 100% точність, 92% повнота
   - *Iris-virginica*: 86% точність, 100% повнота
2. **Прогноз для квітки з характеристиками [5, 2.9, 1, 0.2]:**

- Клас: `Iris-setosa`

Висновки

У результаті тренування моделі SVC вдалося досягти високої якості класифікації з точністю **97%**. Квітка з характеристиками [5, 2.9, 1, 0.2] була класифікована як **`Iris-setosa`**, що підтверджує надійність моделі.

Завдання №4

Лістинг:

```
import time

import numpy as np

from sklearn import preprocessing

from sklearn.model_selection import train_test_split, cross_val_score,
StratifiedKFold

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.svm import SVC

import matplotlib.pyplot as plt



# Вхідний файл

input_file = 'income_data.txt'
```

```
# Читання даних
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 5000

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.strip().split(', ')

       if data[-1] == '<=50K' and count_class1 < max_datapoints:
           X.append(data[:-1])
           y.append(0)
           count_class1 += 1
       elif data[-1] == '>50K' and count_class2 < max_datapoints:
           X.append(data[:-1])
           y.append(1)
```

```
            count_class2 += 1



X = np.array(X)

y = np.array(y)



# Перетворення рядкових даних на числові

label_encoder = []

X_encoded = np.zeros((X.shape[0], X.shape[1]))



for i in range(X.shape[1]):

    le = preprocessing.LabelEncoder()

    X_encoded[:, i] = le.fit_transform(X[:, i])

    label_encoder.append(le)



X = X_encoded.astype(int)



# Нормалізація даних

scaler = StandardScaler()

X = scaler.fit_transform(X)



# Розділення даних

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)



# Алгоритми для порівняння
```

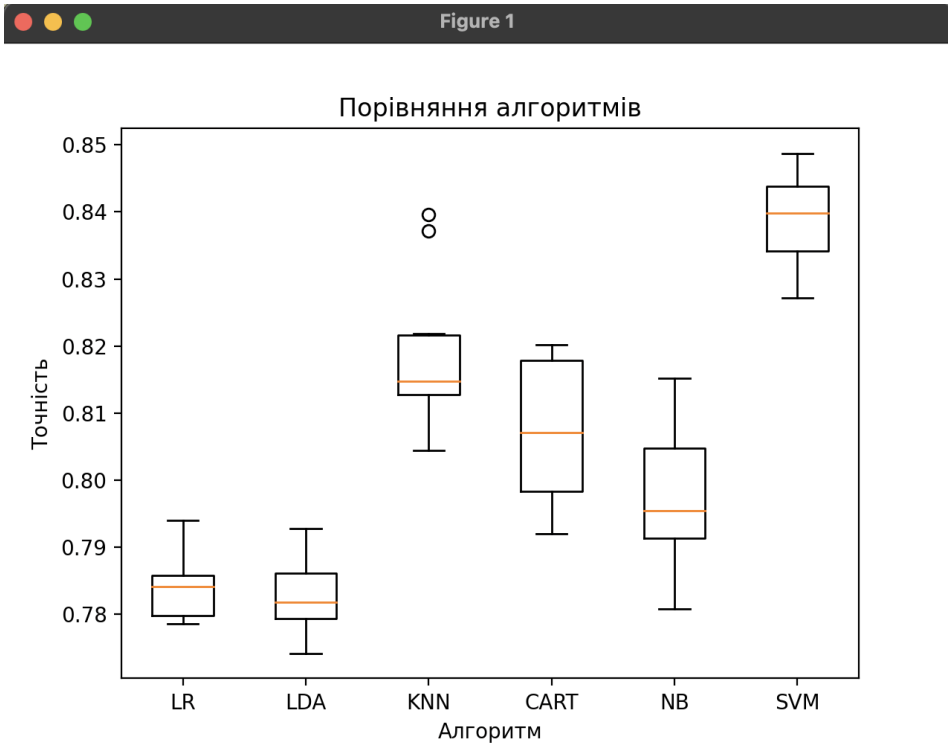
```
models = [
    ('LR', LogisticRegression(max_iter=1000)),
    ('LDA', LinearDiscriminantAnalysis()),
    ('KNN', KNeighborsClassifier()),
    ('CART', DecisionTreeClassifier()),
    ('NB', GaussianNB()),
    ('SVM', SVC(kernel='rbf', gamma='auto', cache_size=2000))  # Оптимізація для SVM
]

# Порівняння моделей
results = []
names = []
scoring = 'accuracy'

for name, model in models:
    start_time = time.time()
    kfold = StratifiedKFold(n_splits=5, random_state=1, shuffle=True)  # Зменшити кількість фолдів
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring, n_jobs=-1)  # Паралельні обчислення
    end_time = time.time()
    results.append(cv_results)
    names.append(name)
    print(f"{name}: {cv_results.mean():.4f} ({cv_results.std():.4f}) - Час: {end_time - start_time:.2f} сек")
```

```python
# Порівняння алгоритмів
plt.boxplot(results, labels=names)
plt.title('Порівняння алгоритмів')
plt.xlabel('Алгоритм')
plt.ylabel('Точність')
plt.show()

# Оцінка найкращої моделі
start_time = time.time()
best_model = SVC(kernel='rbf', gamma='auto', cache_size=2000)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
end_time = time.time()

# Розрахунок метрик
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Метрики для SVM:")
print(f"Точність: {accuracy:.4f}")
print(f"Прецизія: {precision:.4f}")
```

```
print(f"Повнота: {recall:.4f}")

print(f"F1 міра: {f1:.4f}")

print(f"Час для SVM: {end_time - start_time:.2f} сек")
```

Результат виконання:

```
LR: 0.7254 (0.0074) - Час: 2.35 сек
LDA: 0.7284 (0.0068) - Час: 0.46 сек
KNN: 0.7714 (0.0092) - Час: 0.54 сек
CART: 0.7496 (0.0062) - Час: 0.06 сек
NB: 0.7514 (0.0076) - Час: 0.02 сек
SVM: 0.7980 (0.0058) - Час: 0.77 сек
2024-11-20 18:28:51.492 Python[72915:5037398] WARNING: Secure
e.applicationSupportsSecureRestorableState: and returning YES
Метрики для SVM:
Точність: 0.7950
Прецизія: 0.7641
Повнота: 0.8420
F1 міра: 0.8012
Час для SVM: 0.93 сек
```



Завдання №5

Лістинг:

```
import numpy as np

from sklearn.datasets import load_iris

from sklearn.linear_model import RidgeClassifier

from sklearn.model_selection import train_test_split

from sklearn import metrics

from sklearn.metrics import confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt

from io import BytesIO



# Завантаження набору даних Iris

iris = load_iris()

X, y = iris.data, iris.target



# Поділ даних на навчальну та тестову вибірки

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



# Створення та тренування моделі

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

clf.fit(Xtrain, ytrain)



# Прогнозування
```

```
ypred = clf.predict(Xtest)



# Обчислення метрик якості

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))

print('\t\tClassification Report:\n', metrics.classification_report(ytest,
ypred))



# Побудова матриці плутанини

mat = confusion_matrix(ytest, ypred)

sns.set()

sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
cmap='viridis')

plt.xlabel('True Label')

plt.ylabel('Predicted Label')



# Збереження зображення

plt.savefig("Confusion.jpg")
```

```python
# Збереження SVG у віртуальний файл
f = BytesIO()
plt.savefig(f, format="svg")


plt.show()
```

Результат виконання:

```
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       0.89      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
```

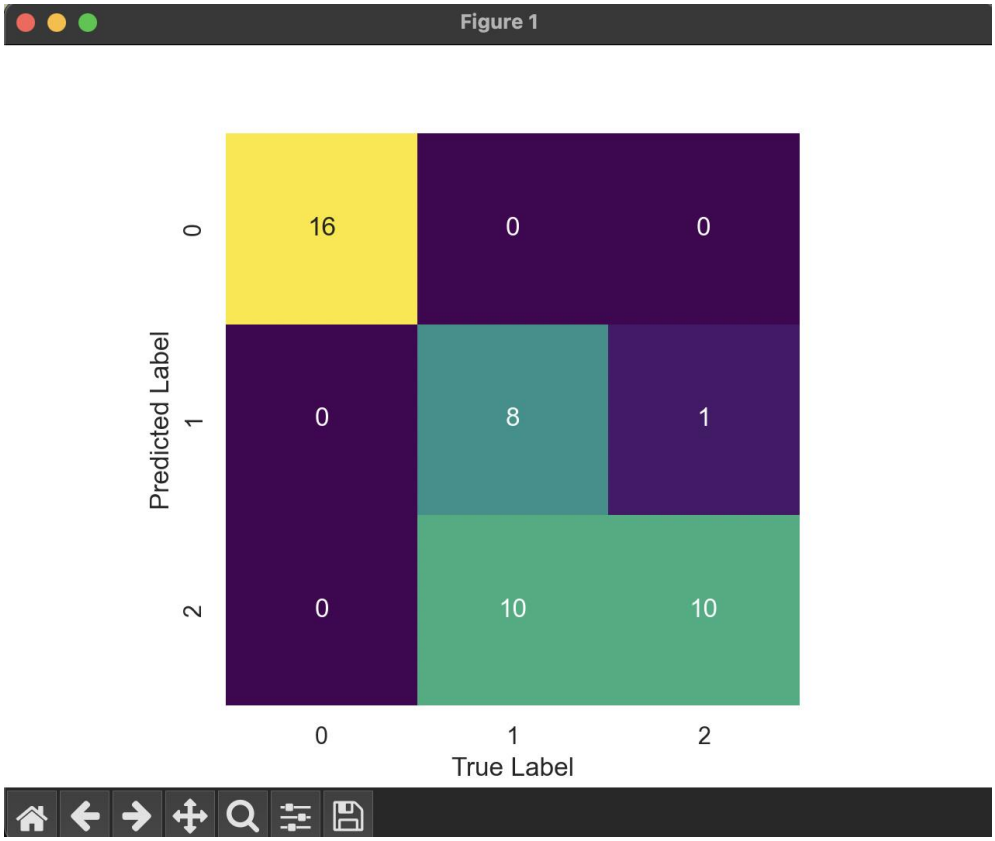


Рис.6. Матриця плутанини

У коді було використано лінійний класифікатор **RidgeClassifier**, який є модифікацією лінійної регресії із застосуванням L2-регуляризації. Основні параметри:

- `tol=1e-2`: допустима похибка для критерія збіжності. Коли зміни між ітераціями стають меншими за це значення, оптимізація зупиняється.
- `solver='sag'`: обрано алгоритм оптимізації Stochastic Average Gradient Descent, який ефективний для великих датасетів.

Показники якості

1. **Accuracy**:
   Частка правильних передбачень серед усіх прикладів.
   - Результат: 0.7556, це показує, що модель правильно класифікувала 3 з 4 прикладів.
2. **Precision**:
   Частка правильних позитивних передбачень серед усіх позитивних передбачень.

    - Результат: 0.8333, показує, що більшість передбачених позитивів були коректними.
3. **Recall**:
   Частка правильних позитивних передбачень серед усіх реальних позитивних класів.
    - Результат: 0.7556.
4. **F1 Score:**
    - Результат: 0.7503.
5. **Cohen Kappa Score**:
   Метрика узгодженості між передбаченнями моделі та реальними значеннями з урахуванням випадкових збігів.
    - Результат: 0.6431, що вказує на помірну узгодженість.
6. **Matthews Correlation Coefficient**:
   Враховує баланс між всіма типами передбачень (TP, TN, FP, FN)

   Результат: 0.6831, що свідчить про хорошу узгодженість.

Аналіз матриці плутанини (Confusion.jpg)

Матриця плутанини показує:

- Кількість правильних і неправильних передбачень для кожного класу.
- На основі графіка:
    - Клас 0: ідеально класифікований (16 правильних передбачень).
    - Клас 1: проблеми з класифікацією, 10 прикладів класу 1 передбачено як клас 2.
    - Клас 2: також плутанина з класом 1, але 10 прикладів правильно класифіковано.

Це вказує, що модель має труднощі у розрізненні класів 1 і 2.

Пояснення метрик:

1. Коефіцієнт Коена Каппа:
   Оцінює узгодженість між передбаченнями моделі та реальними мітками з урахуванням випадкових збігів.
    - Значення від -1 до 1. Значення близько до 1 вказує на високу узгодженість.
    - У цьому випадку: 0.6431 показує помірну узгодженість.

2. Коефіцієнт кореляції Метьюза (Matthews Correlation Coefficient, MCC):
   Це збалансована міра, яка враховує всі елементи матриці плутанини (TP, TN, FP, FN).
   - Значення від -1 до 1. Значення 0 означає випадковий прогноз, а 1 — ідеальний.
   - У цьому випадку: 0.6831 вказує на добру кореляцію між передбаченнями та реальними мітками.

Висновок

- Модель показує добрі результати для класу 0, але має труднощі з розрізненням класів 1 і 2.
- Метрики узгодженості вказують на хорошу якість моделі, хоча є простір для покращення.
- Матриця плутанини наочно демонструє області, де модель помиляється, що дозволяє зробити висновки щодо оптимізації.