ЛАБОРАТОРНА РОБОТА № 2

ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ

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

Git: https://github.com/flekXD/SAI

Завдання 1.

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

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

# Завантаження даних
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000

# Читання 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.strip().split(', ')
        label = data[-1]  # Останній елемент - мітка
        if label == '<=50K' and count_class1 < max_datapoints:
            X.append(data[:-1])  # Додати всі, крім мітки
            y.append(0)  # Мітка для класу '<=50K'
            count_class1 += 1
        elif label == '>50K' and count_class2 < max_datapoints:
            X.append(data[:-1])  # Додати всі, крім мітки
            y.append(1)  # Мітка для класу '>50K'
            count_class2 += 1

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

```
# Перетворення рядкових даних на числові
label_encoder = []
X_encoded = np.empty(X.shape, dtype=object)  # Додано dtype=object для змішаних
даних
for i in range(X.shape[1]):
    if X[0, i].isdigit():
        X_encoded[:, i] = X[:, i].astype(int)
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])

# Визначення вхідних даних і міток
X = X_encoded[:, :-1].astype(int)
y = np.array(y)

# Створення SVM-класифікатора
classifier = OneVsOneClassifier(LinearSVC(random_state=0))

# Розподіл даних на тренувальні та тестові
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)

# Прогнозування результату для тестових даних
y_test_pred = classifier.predict(X_test)

# Обчислення метрик якості
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
f1 = f1_score(y_test, y_test_pred)

# Виведення результатів
print("Звіт про класифікацію:\n", classification_report(y_test, y_test_pred))
print(f"Акуратність: {accuracy:.2f}")
print(f"Точність: {precision:.2f}")
print(f"Повнота: {recall:.2f}")
print(f"F1-міра: {f1:.2f}")

# 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:
        input_data_encoded[i] = label_encoder[count].transform([input_data[i]])[0]
        count += 1

input_data_encoded = np.array(input_data_encoded).reshape(1, -1)
input_data_encoded = input_data_encoded[:, :-1]

# Використання класифікатора для кодованої точки даних та виведення результату
predicted_class = classifier.predict(input_data_encoded)
print("Передбачений клас для тестової точки:", label_encoder[-
1].inverse_transform(predicted_class)[0])
```

Основні зміни:

Виправлення обробки даних: Використано strip() для видалення пробілів у кінці рядка та виправлено порядок зчитування міток.

Виправлення типу масиву: X_encoded тепер має тип object, щоб коректно обробляти змішані дані.

Виправлене кодування тестової точки: Замість використання змінної count, тепер враховуємо всі ознаки під час кодування.

Завдання 2.

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.

```
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 classification_report, accuracy_score, precision_score,
recall_score, f1_score

input_file = 'income_data.txt'
```

```
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 10000

# Читання 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.strip().split(', ')
        label = data[-1]
        if label == '<=50K' and count_class1 < max_datapoints:
            X.append(data[:-1])
            y.append(0)
            count_class1 += 1
        elif label == '>50K' and count_class2 < max_datapoints:
            X.append(data[:-1])
            y.append(1)
            count_class2 += 1

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

label_encoder = []
X_encoded = np.empty(X.shape, dtype=object)
for i in range(X.shape[1]):
    if X[0, i].isdigit():
        X_encoded[:, i] = X[:, i].astype(int)
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])

try:
    X_encoded = X_encoded.astype(int)
except ValueError:
    print("Error: Non-numeric values present after encoding.")

X = X_encoded.astype(int)
y = np.array(y)

# Розподіл даних на тренувальні та тестові
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)

# Створення і навчання класифікатора з поліноміальним ядром
classifier = SVC(kernel='poly', degree=3)  # Зменшення ступеня до 3
classifier.fit(X_train, y_train)
```

```
# Прогнозування результату для тестових даних
y_test_pred = classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
f1 = f1_score(y_test, y_test_pred)

print("\n--- Поліноміальне ядро SVM ---")
print("Звіт про класифікацію:\n", classification_report(y_test, y_test_pred))
print(f"Акуратність: {accuracy:.2f}")
print(f"Точність: {precision:.2f}")
print(f"Повнота: {recall:.2f}")
print(f"F1-міра: {f1:.2f}")
```

LR_2_task_2_1 код для поліноміального ядра

```
PS C:\Users\Fleks\OneDrive\Рабочий стол\123> python lab2_task_2_1.py
>>

--- Поліноміальне ядро SVM ---
Звіт про класифікацію:
              precision    recall  f1-score   support

           0       0.64      0.99      0.78      1968
           1       0.88      0.11      0.20      1230

    accuracy                           0.65      3198
weighted avg       0.73      0.65      0.56      3198

Акуратність: 0.65
Точність: 0.88
Повнота: 0.11
F1-міра: 0.20
```

Виконання коду для поліноміального ядра

```
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 classification_report, accuracy_score, precision_score,
recall_score, f1_score

input_file = 'income_data.txt'

X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 10000
```

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

X = np.array(X)

label_encoder = []
X_encoded = np.empty(X.shape, dtype=object)
for i in range(X.shape[1]):
    if X[0, i].isdigit():
        X_encoded[:, i] = X[:, i].astype(int)
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])

X = X_encoded.astype(int)
y = np.array(y)

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

# Створення і навчання класифікатора з Гаусовим ядром
classifier = SVC(kernel='rbf')
classifier.fit(X_train, y_train)

# Прогнозування результату для тестових даних
y_test_pred = classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
f1 = f1_score(y_test, y_test_pred)

print("\n--- Гауссове ядро SVM ---")
print("Звіт про класифікацію:\n", classification_report(y_test, y_test_pred))
print(f"Акуратність: {accuracy:.2f}")
print(f"Точність: {precision:.2f}")
print(f"Повнота: {recall:.2f}")
```

```
print(f"F1-міра: {f1:.2f}")
```

LR_2_task_2_2 код для Гаусового ядра

```
PS C:\Users\Fleks\OneDrive\Рабочий стол\123> python lab2_task_2_2.py
>>

--- Гауссове ядро SVM ---
Звіт про класифікацію:
              precision    recall  f1-score   support

           0       0.65      1.00      0.79      1968
           1       0.98      0.14      0.25      1230

    accuracy                           0.67      3198
   macro avg       0.81      0.57      0.52      3198
weighted avg       0.78      0.67      0.58      3198

Акуратність: 0.67
Точність: 0.98
Повнота: 0.14
F1-міра: 0.25
```

Виконання коду для Гаусового ядра

```
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 classification_report, accuracy_score, precision_score,
recall_score, f1_score

input_file = 'income_data.txt'

X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 10000

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

```
            count_class2 += 1

X = np.array(X)

label_encoder = []
X_encoded = np.empty(X.shape, dtype=object)
for i in range(X.shape[1]):
    if X[0, i].isdigit():
        X_encoded[:, i] = X[:, i].astype(int)
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])

X = X_encoded.astype(int)
y = np.array(y)

# Розподіл даних на тренувальні та тестові
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)

# Створення і навчання класифікатора з сигмоїдальним ядром
classifier = SVC(kernel='sigmoid')
classifier.fit(X_train, y_train)

# Прогнозування результату для тестових даних
y_test_pred = classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
f1 = f1_score(y_test, y_test_pred)

print("\n--- Сигмоїдальне ядро SVM ---")
print("Звіт про класифікацію:\n", classification_report(y_test, y_test_pred))
print(f"Акуратність: {accuracy:.2f}")
print(f"Точність: {precision:.2f}")
print(f"Повнота: {recall:.2f}")
print(f"F1-міра: {f1:.2f}")
```

LR_2_task_2_3 код для сигмоїдного ядра

```
PS C:\Users\Fleks\OneDrive\Рабочий стол\123> python lab2_task_2_3.py
>>

--- Сигмоїдальне ядро SVM ---
Звіт про класифікацію:
              precision    recall  f1-score   support

           0       0.60      0.61      0.61      1968
           1       0.36      0.35      0.36      1230

    accuracy                           0.51      3198
   macro avg       0.48      0.48      0.48      3198
weighted avg       0.51      0.51      0.51      3198

Акуратність: 0.51
Точність: 0.36
Повнота: 0.35
F1-міра: 0.36
```

Виконання коду для сигмоїдного ядра

Найкраще для завдання класифікації спрацювало гауссове ядро (RBF), оскільки воно досягло найвищої загальної точності (0.67). Це свідчить про те, що гауссове ядро краще обробляє нелінійні особливості у даних. Однак низька повнота для класу "1" вказує на потребу подальшого налаштування гіперпараметрів або використання інших методів, таких як зважування класів або балансування даних.

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.

Крок 1:

```python
from sklearn.datasets import load_iris

# Завантаження набору даних
iris_dataset = load_iris()

# Ключі об'єкта iris_dataset
print("Ключі iris_dataset: \n{}".format(iris_dataset.keys()))

# Опис набору даних
print("Опис набору даних:\n", iris_dataset['DESCR'][:193] + "\n...")
```

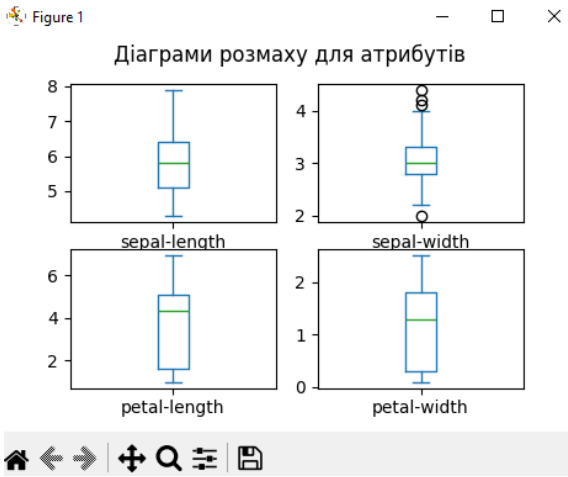
```
# Назви відповідей (класів)
print("Назви відповідей: {}".format(iris_dataset['target_names']))

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

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

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

# Виведення значень ознак для перших п'яти прикладів
print("Перші п'ять прикладів:\n{}".format(iris_dataset['data'][:5]))

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

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

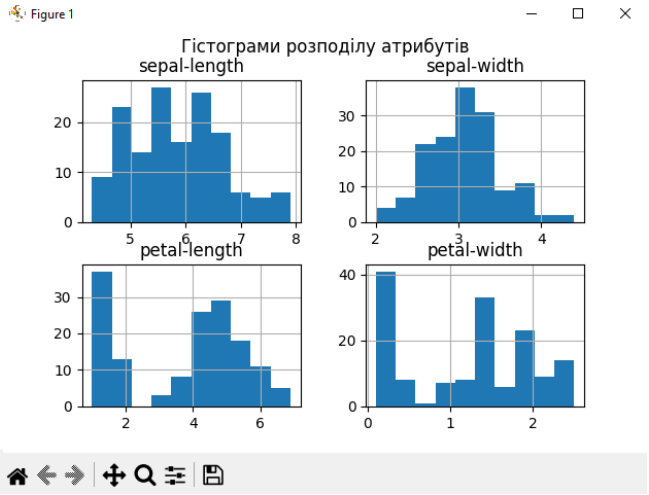
```
PS D:\123> python lab2_3.py
Ключі iris_dataset:
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
Опис набору даних:
 .. _iris_dataset:

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)
Перші п'ять прикладів:
[[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]]
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 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 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]
```

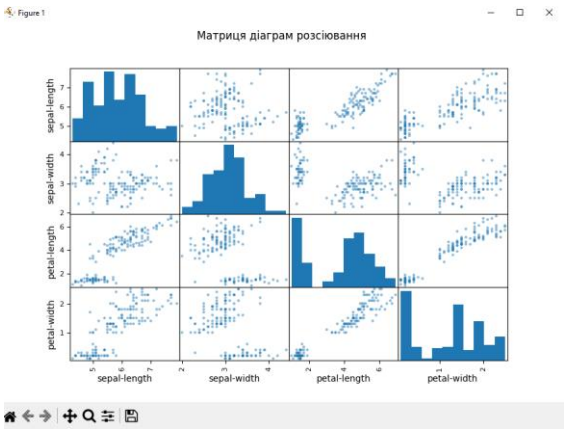
Завантаження та вивчення даних

```
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("Розмір датасету:", dataset.shape)

print("\nПерші 20 рядків:")
print(dataset.head(20))

print("\nСтатистичне зведення:")
print(dataset.describe())
```

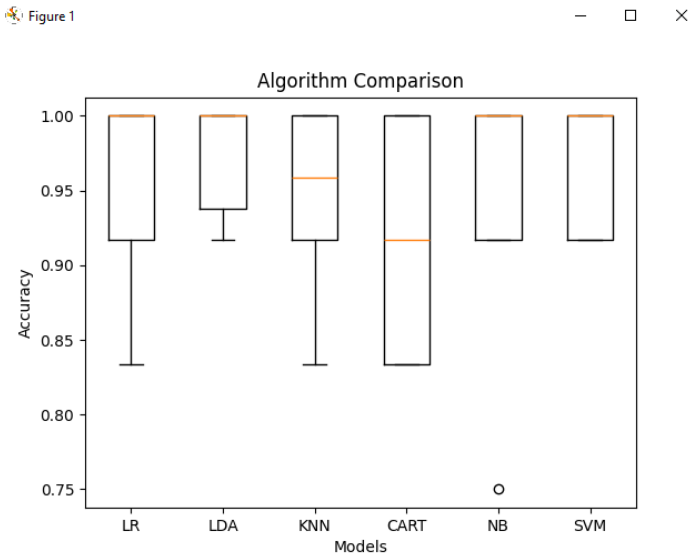
```
print("\nРозподіл за класами:")
print(dataset.groupby('class').size())

# Діаграма розмаху для кожного атрибута
dataset.plot(kind='box', subplots=True, layout=(2, 2), sharex=False, sharey=False,
figsize=(10, 6))
plt.suptitle("Діаграми розмаху для атрибутів")
plt.show()

# Гістограма для кожного атрибута
dataset.hist(figsize=(10, 6))
plt.suptitle("Гістограми розподілу атрибутів")
plt.show()

# Матриця діаграм розсіювання
scatter_matrix(dataset, figsize=(12, 8), diagonal='hist')
plt.suptitle("Матриця діаграм розсіювання")
plt.show()
```







Крок 3-4:



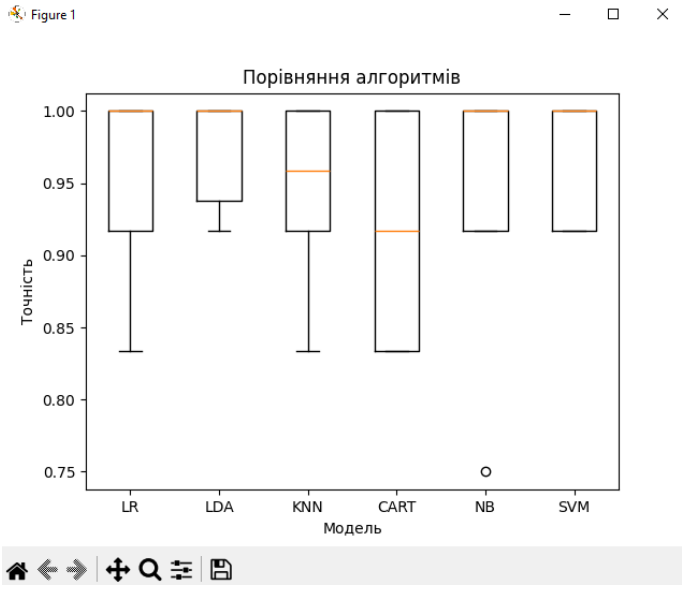
Найкращим виявився “Наївний баєсовський класифікатор” (NB) з найбішою точністю та мінімальним відхиленням

```
LR: Середня точність = 0.9583, Стандартне відхилення = 0.0559
LDA: Середня точність = 0.9750, Стандартне відхилення = 0.0382
KNN: Середня точність = 0.9500, Стандартне відхилення = 0.0553
CART: Середня точність = 0.9167, Стандартне відхилення = 0.0745
NB: Середня точність = 0.9500, Стандартне відхилення = 0.0764
SVM: Середня точність = 0.9667, Стандартне відхилення = 0.0408
```

Крок 5-8

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
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

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_validation, y_train, y_validation = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

# КРОК 2: Побудова та оцінка моделей
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')))

# Оцінка моделей
results = []
names = []
print("Результати оцінки моделей (точність):")
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(f"{name}: {cv_results.mean():.4f} ({cv_results.std():.4f})")

# Порівняння алгоритмів на графіку
plt.boxplot(results, labels=names)
plt.title('Порівняння алгоритмів')
plt.xlabel('Модель')
plt.ylabel('Точність')
plt.show()

# КРОК 6: Створення прогнозу на тестовій вибірці
model = SVC(gamma='auto')
model.fit(X_train, y_train)  # Навчаємо модель
predictions = model.predict(X_validation)  # Прогноз на тестовій вибірці

# КРОК 7: Оцінка якості моделі
print("\nТочність моделі на тестовій вибірці:", accuracy_score(y_validation,
predictions))
print("\nМатриця помилок:")
print(confusion_matrix(y_validation, predictions))
print("\nЗвіт про класифікацію:")
print(classification_report(y_validation, predictions))

# КРОК 8: Передбачення для нових даних
X_new = np.array([[5, 2.9, 1, 0.2]])
print("\nФорма масиву X_new:", X_new.shape)

new_prediction = model.predict(X_new)
print("\nПрогнозований клас для нових даних:", new_prediction[0])
print("Спрогнозований сорт ірису:", iris.target_names[new_prediction[0]])
```



Вивид графіків

```
Точність моделі на тестовій вибірці: 0.9666666666666667

Матриця помилок:
[[10  0  0]
 [ 0 10  0]
 [ 0  1  9]]

Звіт про класифікацію:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.91      1.00      0.95        10
           2       1.00      0.90      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30


Форма масиву X_new: (1, 4)

Прогнозований клас для нових даних: 0
Спрогнозований сорт ірису: setosa
PS D:\123>
```

Вивід даних в консоль

Висновок:

Якість класифікації:

Точність моделі: 96.67%, що свідчить про високу ефективність класифікації.

Матриця помилок: більшість класів класифікуються правильно, з деякими помилками між класами `1` (versicolor) та `2` (virginica).

Звіт про класифікацію: висока точність і повний збіг для класу `0`, хороші результати для інших класів.

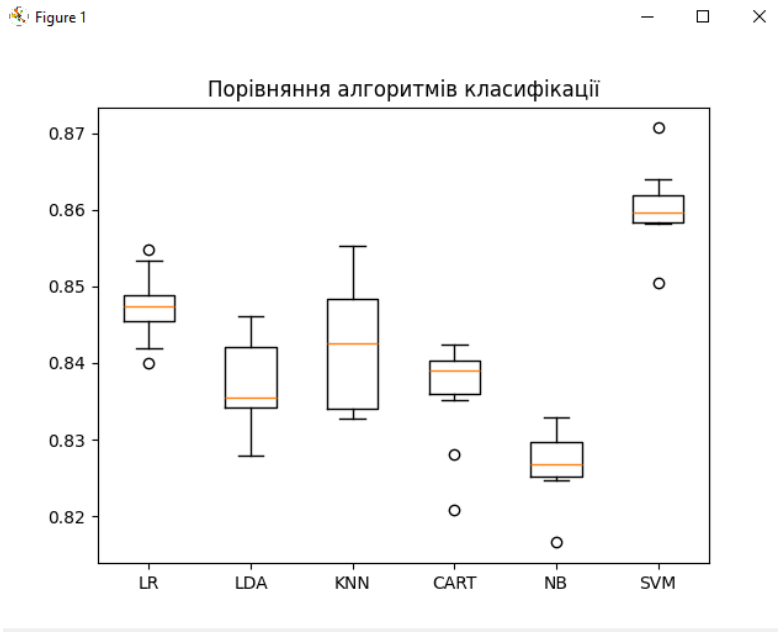
Прогноз для нових даних:

Квітка з новими характеристиками була класифікована як setosa.

Завдання 2.4 Порівняння якості класифікаторів для набору даних завдання 2.1

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score,
StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix, 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 matplotlib import pyplot as plt

data_path = "income_data.txt"
columns = ['age', 'workclass', 'education', 'education-num', 'marital-status',
           'occupation', 'relationship', 'race', 'sex', 'hours-per-week',
           'native-country', 'income']
dataset = pd.read_csv(data_path, header=None, names=columns)


dataset = pd.get_dummies(dataset, drop_first=True)

X = dataset.drop('income_ >50K', axis=1)  # Вхідні ознаки
y = dataset['income_ >50K']  # Мітки класів

X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)

models = []
models.append(('LR', 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 = []

print("\nРезультати моделей:")
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()))

plt.boxplot(results, labels=names)
plt.title('Порівняння алгоритмів класифікації')
plt.show()

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

print("\nТочність:", accuracy_score(Y_validation, predictions))
print("Матриця помилок:")
print(confusion_matrix(Y_validation, predictions))
print("\nЗвіт про класифікацію:")
print(classification_report(Y_validation, predictions))
```



```
Результати моделей:
LR: 0.847377 (0.004315)
LDA: 0.837078 (0.005954)
KNN: 0.842107 (0.007706)
CART: 0.836504 (0.006518)
NB: 0.826826 (0.004357)
SVM: 0.860215 (0.004846)
```

З результатів видно, що SVM має найвищу точність (0.8602), що на 1.5% вищий за точність логістичної регресії (0.8474) та метод k-найближчих сусідів (0.8421). Всі моделі мають подібну варіативність (стандартне відхилення не перевищує 0.01), що свідчить про стабільність цих моделей на тестових даних.Проте , дана модель найповільніша.

Пояснення вибору найкращої моделі:

Вибір SVM як найкращої моделі для цієї задачі обґрунтовано кількома факторами: Найвища точність: SVM продемонстрував найвищу точність, що є головним критерієм для порівняння класифікаторів. Стабільність: Стандартне відхилення для SVM (0.0048) є одним з найменших, що свідчить про стабільність моделі.

Підходить для високовимірних даних: SVM добре працює з високовимірними даними, що може бути корисним, якщо в майбутньому з’являться нові змінні.

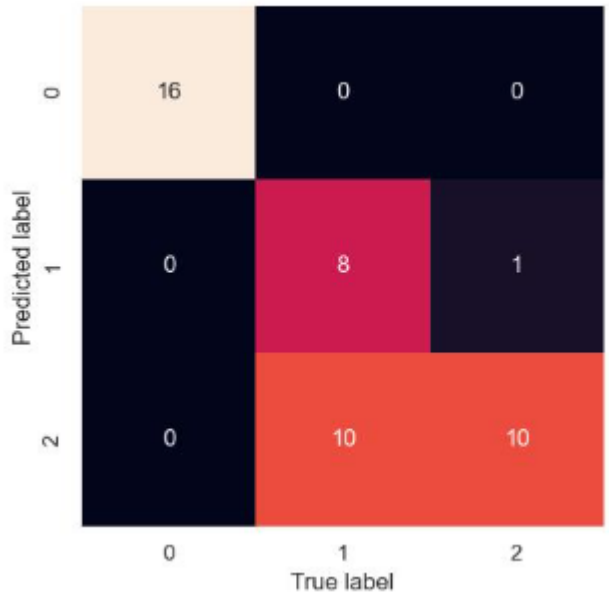
Висновок :

Найкраща модель: SVM є найкращим вибором для цієї задачі на основі точності та стабільності результатів. Рекомендація: Хоча інші моделі також можуть бути корисними, SVM дає найкращі результати для поточного набору даних.

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

```python
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
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
```

```
# Розподіл на тренувальні та тестові дані
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.3,
random_state=0)

# Ініціалізація і тренування класифікатора Ridge
clf = RidgeClassifier(tol=1e-2, solver="sag")
clf.fit(Xtrain, ytrain)

# Прогнозування на тестових даних
ypred = clf.predict(Xtest)

# Виведення показників якості класифікації
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))

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

# Матриця плутанини
mat = confusion_matrix(ytest, ypred)
sns.set()
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('True label')
plt.ylabel('Predicted label')

# Збереження зображення у файл
plt.savefig("Confusion.jpg")

# Збереження зображення у форматі SVG
f = BytesIO()
plt.savefig(f, format="svg")
```



Пояснення:

Завантаження та розподіл даних:

Данні Iris завантажуються за допомогою load_iris().

Використовуємо train_test_split() для розподілу даних на тренувальні та тестові набори. 70% даних йде на тренування, а 30% — на тестування.

Ініціалізація та тренування класифікатора:

Класифікатор Ridge ініціалізується з параметрами:

tol=1e-2: поріг для завершення ітерацій.

solver="sag": використовується метод Stochastic Average Gradient.

Модель тренується за допомогою clf.fit(Xtrain, ytrain).

Прогнозування та оцінка моделі:

Прогнозування здійснюється за допомогою clf.predict(Xtest).

Для оцінки точності моделі використовуються різні метрики:

Accuracy: частка правильних прогнозів.

Precision: точність класифікації.

Recall: повнота класифікації.

F1 Score: середнє гармонійне точності та повноти.

Cohen Kappa: вимірює узгодженість між класифікатором та реальними мітками.

Matthews Correlation Coefficient (MCC): вимірює кореляцію між реальними та прогнозованими мітками.

Матриця плутанини:

Використовуємо confusion_matrix() для створення матриці плутанини.

За допомогою бібліотеки Seaborn sns.heatmap() будується теплове зображення цієї матриці для візуалізації.

Збереження результатів:

Зображення матриці плутанини зберігається у файл Confusion.jpg.

Також зберігається векторне зображення у форматі SVG.

Пояснення використаних показників:

Cohen Kappa вимірює узгодженість класифікаційних результатів, зважаючи на випадкові ймовірності для кожного класу.

Matthews Correlation Coefficient (MCC) забезпечує більш стабільну оцінку ефективності класифікатора, особливо у разі незбалансованих класів, враховуючи всі чотири категорії: True Positive (TP), True Negative (TN), False Positive (FP) і False Negative (FN).

Завдяки цим метрикам можна отримати комплексну оцінку якості моделі, яка дозволяє краще зрозуміти її ефективність для класифікації даних Iris.

Висновок:

У роботі були порівняні різні методи класифікації, зокрема SVM, LDA, KNN, LR, CART, NB, та Ridge. Найвищу точність показав метод опорних векторів (SVM). Вибір найкращого методу залежить від конкретної задачі, проте для даних Iris SVM виявився найбільш ефективним.