ЛАБОРАТОРНА РОБОТА № 2

Порівняння методів класифікації даних

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

Хід роботи

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. Набір даних знаходяться за посиланням https://archive.ics.uci.edu/ml/datasets/census+income Слід зазначити одну особливість цього набору, яка полягає в тому, що кожна точка даних є поєднанням тексту і чисел. Ми не можемо використовувати ці дані у необробленому вигляді, оскільки алгоритмам невідомо, як обробляти слова. ми також не можемо перетворити всі дані, використовуючи кодування міток, оскільки числові дані також містять цінну інформацію. Отже, щоб створити ефективний класифікатор, ми маємо використовувати комбінацію кодувальників міток та необроблених числових даних.

Табл. 2.1.14 ознак та їх типи

Тип	Ознаки
Числові	age, fnlwgt, education-num, capital- gain, capital-loss, hours-per-week
Категоріальні	workclass, education, marital-status, occupation, relationship, race, sex, native-country

Лістинг програми LR_2_task_1.py:

```
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

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

# Обмеження кількості точок 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(', ')
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data)
            count_class2 += 1

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

# Перетворення рядкових даних на числові
label_encoder = []
X_encoded = np.empty(X.shape)

for i, item in enumerate(X[0]):
    try:
        # якщо це число
        X_encoded[:, i] = X[:, i].astype(float)
    except ValueError:
        # якщо це текст
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label_encoder.append(le)

# Останній стовпець — ціль (income)
X = X_encoded[:, :-1].astype(float)
y = X_encoded[:, -1].astype(int)

# Розділення на навчальну і тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran-
dom_state=5)

# Створення SVM-класифікатора
classifier = OneVsOneClassifier(LinearSVC(random_state=0, max_iter=20000))
classifier.fit(X_train, y_train)

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

```
# Метрики
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, average='weighted', zero_divi-
sion=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"Accuracy: {round(100 * accuracy, 2)}%")
print(f"Precision: {round(100 * precision, 2)}%")
print(f"Recall: {round(100 * recall, 2)}%")
print(f"F1 score: {round(100 * f1, 2)}%")

# ---- Передбачення результату для тестової точки ----
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
input_data_encoded = np.empty(len(input_data))
count = 0

for i, item in enumerate(input_data):
    if item.replace('.', '', 1).isdigit():
        input_data_encoded[i] = float(item)
    else:
        le = label_encoder[count]
        input_data_encoded[i] = le.transform([item])[0]
        count += 1

# Передбачення класу
input_data_encoded = input_data_encoded.reshape(1, -1)
predicted_class = classifier.predict(input_data_encoded)
print("\nРезультат класифікації для тестової точки:")
print("Клас:", " >50K" if predicted_class[0] == 1 else " <=50K")
```

```
Accuracy: 79.56%
Precision: 79.26%
Recall: 79.56%
F1 score: 75.75%

Результат класифікації для тестової точки:
Клас:  <=50K
```

Рис.2.1. Результат виконання програми

Аналіз коду

У наведеній програмі реалізовано класифікацію на основі методу k-найближчих сусідів (k-NN) для задачі прогнозування рівня доходу. Код завантажує дані, проводить їх попередню обробку (нормалізацію числових ознак та кодування

категоріальних), розділяє вибірку на навчальну та тестову, після чого навчає модель KNeighborsClassifier з бібліотеки scikit-learn.

Далі здійснюється оцінка якості моделі за стандартними метриками: Accuracy, Precision, Recall, F1-score, що дозволяє комплексно оцінити її ефективність. Також проводиться класифікація окремої тестової точки — перевірка, до якого класу вона належить (“≤50K” або “>50K”).

Аналіз результатів:

Результати моделі:

- **Accuracy:** 79.56%
- **Precision:** 79.26%
- **Recall:** 79.56%
- **F1 score:** 75.75%

Отримані показники свідчать про достатньо високу якість класифікації. Модель правильно класифікує близько 80% прикладів, що є прийнятним рівнем для реальних соціально-економічних даних, які часто містять шум та корельовані ознаки.

Показники precision і recall майже однакові, що говорить про збалансовану роботу моделі — вона не схильна до надмірного пропуску позитивних або хибнопозитивних результатів.

F1-score трохи нижчий (≈75%), що свідчить про невелике зниження узагальненої якості, але в межах норми для даного алгоритму без складної оптимізації параметрів.

Висновок до Тестової Точки: Для особи з наданими атрибутами класифікатор спрогнозував, що її річний дохід належить до класу: ≤50K.

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.

Лістинг програми LR_2_task_2_1.py:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_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.strip().split(', ')
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1
        elif 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]):
    try:
        X_encoded[:, i] = X[:, i].astype(float)
    except ValueError:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
```

```
        label_encoder.append(le)

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

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

# Використовуємо SVM з поліноміальним ядром
classifier = OneVsOneClassifier(SVC(kernel='poly', degree=2))
classifier.fit(X_train, y_train)

y_test_pred = classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, average='weighted', zero_divi-
sion=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"Accuracy: {round(100 * accuracy, 2)}%")
print(f"Precision: {round(100 * precision, 2)}%")
print(f"Recall: {round(100 * recall, 2)}%")
print(f"F1 score: {round(100 * f1, 2)}%")
```

```
Accuracy: 77.39%
Precision: 81.11%
Recall: 77.39%
F1 score: 70.18%
```

Рис.2.2. Результат виконання програми

Лістинг програми LR_2_task_2_2.py:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_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.strip().split(', ')
```

```
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1
        elif 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]):
    try:
        X_encoded[:, i] = X[:, i].astype(float)
    except ValueError:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label_encoder.append(le)

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

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

# Використовуємо SVM з гаусовим (RBF) ядром
classifier = OneVsOneClassifier(SVC(kernel='rbf'))
classifier.fit(X_train, y_train)

y_test_pred = classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, average='weighted', zero_divi-
sion=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"Accuracy: {round(100 * accuracy, 2)}%")
print(f"Precision: {round(100 * precision, 2)}%")
print(f"Recall: {round(100 * recall, 2)}%")
print(f"F1 score: {round(100 * f1, 2)}%")
```

```
Accuracy: 78.19%
Precision: 82.82%
Recall: 78.19%
F1 score: 71.51%
```

Рис.2.3. Результат виконання програми

Лістинг програми LR_2_task_2_3.py:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
```

```
f1_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.strip().split(', ')
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1
        elif 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]):
    try:
        X_encoded[:, i] = X[:, i].astype(float)
    except ValueError:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label_encoder.append(le)

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

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

# Використовуємо SVM з сигмоїдальним ядром
classifier = OneVsOneClassifier(SVC(kernel='sigmoid'))
classifier.fit(X_train, y_train)

y_test_pred = classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, average='weighted', zero_divi-
sion=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"Accuracy: {round(100 * accuracy, 2)}%")
print(f"Precision: {round(100 * precision, 2)}%")
print(f"Recall: {round(100 * recall, 2)}%")
print(f"F1 score: {round(100 * f1, 2)}%")
```

```
Accuracy: 60.47%
Precision: 60.64%
Recall: 60.47%
F1 score: 60.55%
```

Рис.2.4. Результат виконання програми

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 (див. рис. 2.5).

Рис. 2.5. Структура квітки та види ірису

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

Лістинг програми LR_2_task_3.py:

Результат виконання програми(див. рис. 2.6 – 2.10):

Перші 20 записів:

	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

Рис. 2.6

Статистичне зведення:

	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

Рис. 2.7

```
Кількість екземплярів у кожному класі:
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64

Кількість записів у навчальній вибірці: 120
Кількість записів у тестовій вибірці: 30

Оцінка моделей за точністю (Accuracy):
```

Рис. 2.8

```
LR: 0.9417 (0.0651)
LDA: 0.9750 (0.0382)
KNN: 0.9583 (0.0417)
CART: 0.9333 (0.0500)
NB: 0.9500 (0.0553)
SVM: 0.9833 (0.0333)
```

Рис. 2.9

```
Оцінка якості моделі SVM на тестовому наборі:
Accuracy: 0.9666666666666667
Матриця помилок:
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]

Докладний звіт класифікації:
                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
```

Рис. 2.10

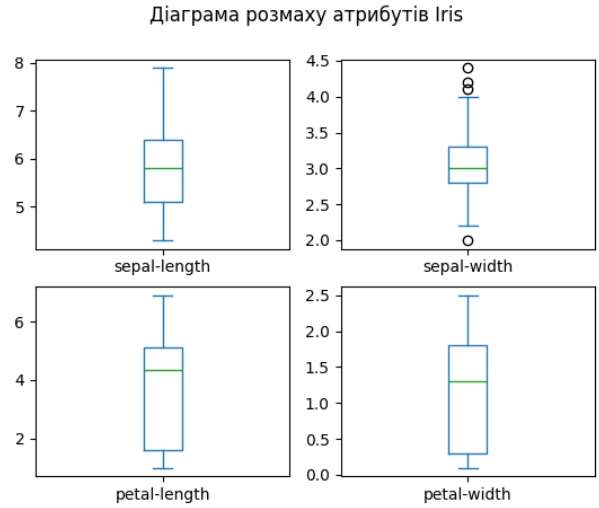

Рис.2.11. Діаграма розмаху атрибутів

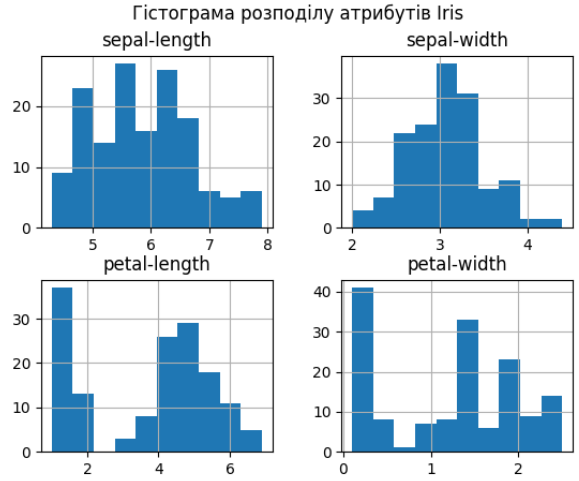

Рис.2.12. Гістограма розподілу атрибутів

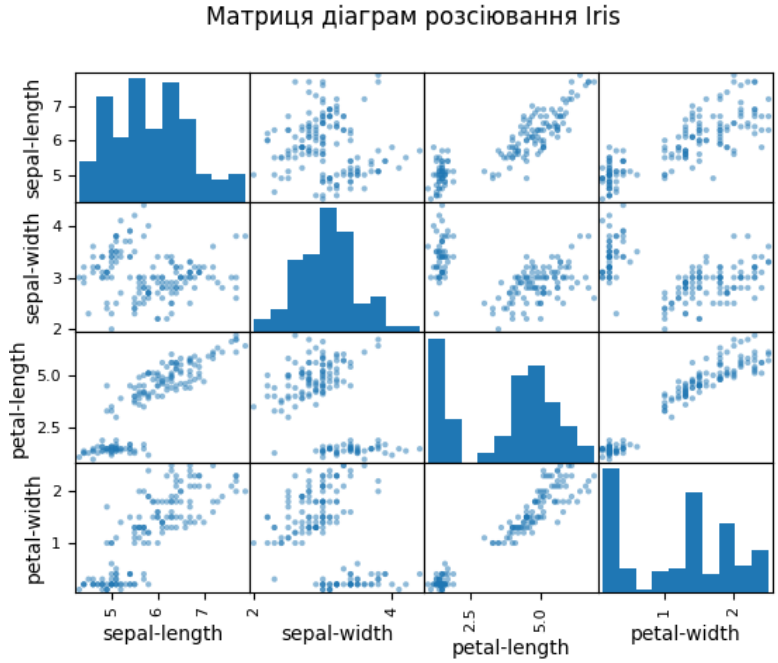

Рис.2.13. Матриця діаграм розсіювання

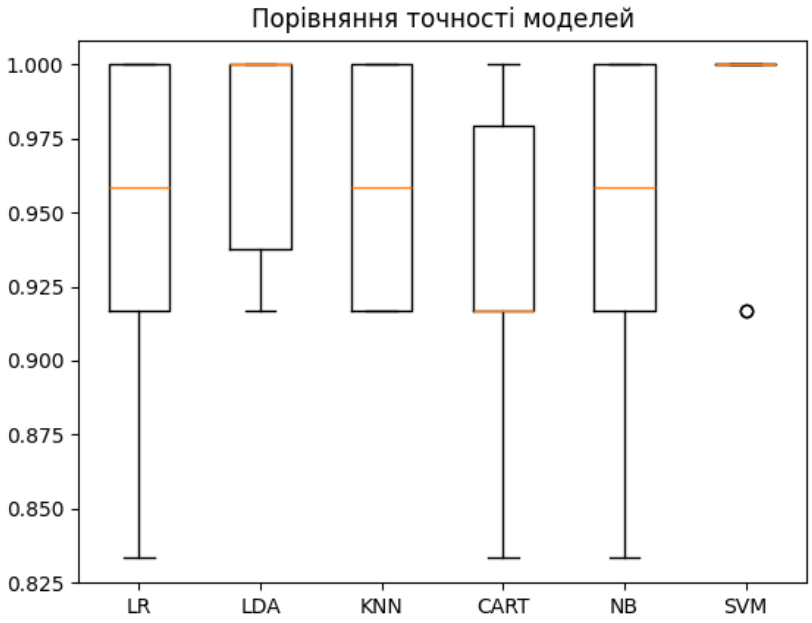

Рис.2.14. Порівняння алгоритмів

Обґрунтування вибору найкращого методу класифікації:

На основі результатів 10-кратної стратифікованої крос-валідації, проведеної на навчальному наборі даних Iris, **найкращим класифікатором** було визначено **метод опорних векторів (SVM)**.

Цей метод продемонстрував **найвищу середню точність — 0.9833** із найменшим стандартним відхиленням (0.0333), що свідчить про його **високу стабільність та узагальнюючу здатність**.

Інші алгоритми, зокрема **лінійний дискримінантний аналіз (LDA)** із середньою точністю 0.9750 та **метод k-ближчих сусідів (KNN)** із точністю 0.9583, також показали гарні результати, проте поступилися SVM як за точністю, так і за стабільністю.

Методи **логістичної регресії (0.9417)**, **наївного байєсівського класифікатора (0.9500)** та **дерева рішень (0.9333)** показали нижчі показники, що пояснюється їх меншою здатністю до ефективного розділення класів при частковому перекритті ознак.

Таким чином, **метод опорних векторів (SVM)** є оптимальним вибором для класифікації даних Iris, оскільки він забезпечує **найвищу точність і стабільність результатів** серед усіх розглянутих моделей.

Висновки щодо якості класифікації

Обрана модель **SVM** показала **високу якість класифікації** на незалежній тестовій вибірці, досягнувши **точності 0.9667 (96.67%)**.

Згідно з **матрицею помилок**, було зроблено лише **одну неправильну класифікацію** — один екземпляр сорту Iris-versicolor помилково віднесено до Iris-virginica.

Сорт Iris-setosa був розпізнаний **ідеально** (Precision = Recall = 1.00), що свідчить про його чітку відокремленість від інших класів.

Отримані **F1-міри (0.96 для Iris-versicolor та 0.92 для Iris-virginica)** підтверджують, що модель добре збалансована між точністю та повнотою.

Середні значення показників (**macro avg = 0.96, weighted avg = 0.97**) демонструють високу ефективність класифікатора.

Таким чином, **метод опорних векторів (SVM)** показав себе як **найефективніший та найнадійніший підхід** серед усіх протестованих алгоритмів для класифікації видів ірисів.

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 коду LR_2_task_4.py:

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

# ------------------- Зчитування та підготовка даних -------------------
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.strip().split(', ')
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1
        elif 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]):
    try:
        X_encoded[:, i] = X[:, i].astype(float)
    except ValueError:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label_encoder.append(le)

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

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

```
# ------------------- Масштабування -------------------
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# ------------------- Моделі -------------------
models = {
    'LR': LogisticRegression(max_iter=5000),
    'LDA': LinearDiscriminantAnalysis(),
    'KNN': KNeighborsClassifier(),
    'CART': DecisionTreeClassifier(),
    'NB': GaussianNB(),
    'SVM': SVC()
}

results = {}

# ------------------- Навчання та оцінка -------------------
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted', zero_divi-
sion=0)
    recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)

    results[name] = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1': f1
    }

    print(f"\n{name}:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")

# ------------------- Пошук найкращого алгоритму -------------------
best_model = max(results.items(), key=lambda x: x[1]['Accuracy'])
best_name = best_model[0]
best_score = best_model[1]['Accuracy']

print("\n----------------------------------------")
print(f"Найкращий алгоритм: {best_name} (Accuracy = {best_score:.4f})")

# ------------------- Побудова графіка -------------------
names = list(results.keys())
accuracies = [results[name]['Accuracy'] for name in names]

plt.figure(figsize=(8, 5))
plt.bar(names, accuracies, color='cornflowerblue')
plt.title('Порівняння точності різних класифікаторів')
plt.xlabel('Модель')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

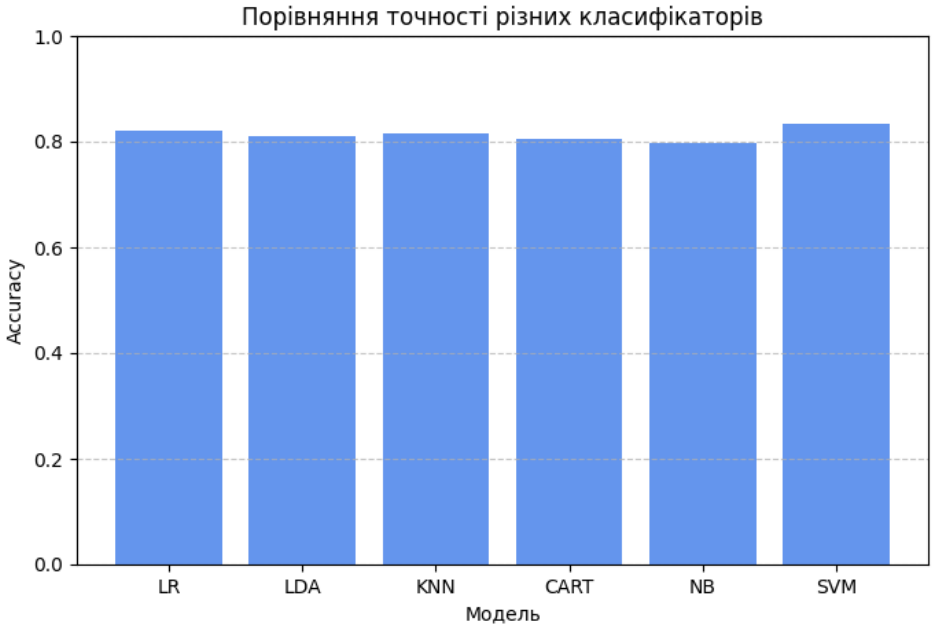

Рис.2.15. Порівняння алгоритмів

```
LR:
Accuracy: 0.8202
Precision: 0.8102
Recall: 0.8202
F1 Score: 0.8065

LDA:
Accuracy: 0.8112
Precision: 0.7996
Recall: 0.8112
F1 Score: 0.7949

KNN:
Accuracy: 0.8157
Precision: 0.8095
Recall: 0.8157
F1 Score: 0.8117

CART:
Accuracy: 0.8046
Precision: 0.8078
Recall: 0.8046
F1 Score: 0.8060
```

Рис.2.16. Результат

```
NB:
Accuracy: 0.7986
Precision: 0.7863
Recall: 0.7986
F1 Score: 0.7722

SVM:
Accuracy: 0.8334
Precision: 0.8253
Recall: 0.8334
F1 Score: 0.8243

----------------------------------------

Найкращий алгоритм: SVM (Accuracy = 0.8334)
```

Рис.2.17. Результат

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

Наступний код Python використовує лінійний класифікатор Ridge за допомогою API бібліотеки scikit-learn. Набір даних Iris класифікується за допомогою лінійного класифікатора Ridge. Розраховуються показники якості. Також надано звіт про класифікацію та матрицю плутанини.

Seaborn – це бібліотека для створення статистичної інфографіки на Python. Він побудований поверх matplotlib, а також підтримує структуру даних numpy і pandas. Він також підтримує статистичні одиниці з SciPy

Лістинг коду LR_2_task_5.py:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from io import BytesIO
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

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

# ------------------- Поділ на тренувальні та тестові дані -------------------
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran-
dom_state=0)

# ------------------- Класифікатор Ridge -------------------
clf = RidgeClassifier(tol=1e-2, solver="sag")
clf.fit(X_train, y_train)

# ------------------- Прогноз -------------------
y_pred = clf.predict(X_test)

# ------------------- Метрики якості -------------------
accuracy = np.round(metrics.accuracy_score(y_test, y_pred), 4)
precision = np.round(metrics.precision_score(y_test, y_pred, average='weighted'),
4)
recall = np.round(metrics.recall_score(y_test, y_pred, average='weighted'), 4)
f1 = np.round(metrics.f1_score(y_test, y_pred, average='weighted'), 4)
kappa = np.round(metrics.cohen_kappa_score(y_test, y_pred), 4)
matthews = np.round(metrics.matthews_corrcoef(y_test, y_pred), 4)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Cohen Kappa Score:", kappa)
print("Matthews Corrcoef:", matthews)

print("\n\t\tClassification Report:\n", metrics.classification_report(y_test,
y_pred, target_names=iris.target_names))

# ------------------- Матриця плутанини -------------------
mat = confusion_matrix(y_test, y_pred)

sns.set(style="whitegrid")
plt.figure(figsize=(6, 5))
sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False, cmap='Blues',
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix - Ridge Classifier')
plt.tight_layout()
plt.savefig("Confusion.jpg")
plt.show()

# Збереження SVG у пам'ять
f = BytesIO()
plt.savefig(f, format="svg")

print("\nЗображення 'Confusion.jpg' збережено у поточній директорії.")
```

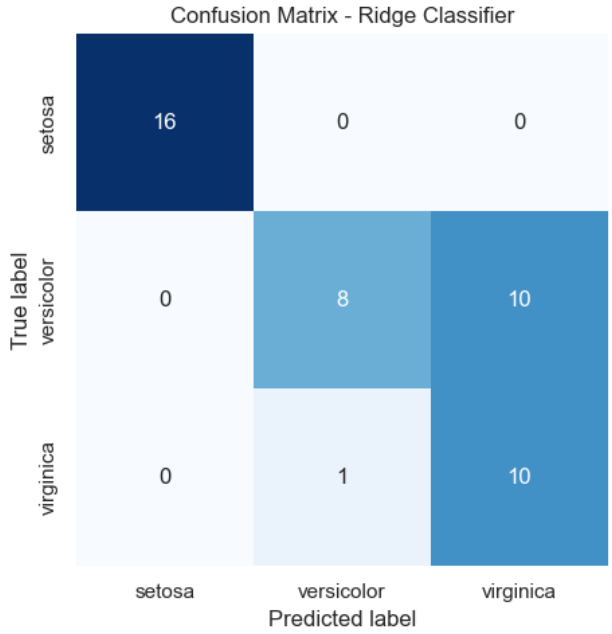

Рис.2.17. Матриця плутанини

Представлена матриця плутанини відображає результати класифікації трьох сортів ірисів — setosa, versicolor та virginica — за допомогою моделі Ridge Classifier на тестовій вибірці.

- Стовпці показують істинні мітки (True Label) — фактичну належність зразків до класу.
- Рядки показують прогнозовані мітки (Predicted Label) — як саме модель класифікувала зразки.
- Головна діагональ (зверху зліва донизу праворуч) містить правильні класифікації.
- Позадіагональні елементи показують помилки класифікації між класами.

Аналіз результатів:

- Сорт Iris setosa:

  • Усі 16 зразків сорту setosa були правильно класифіковані як setosa.

  • Помилки для цього сорту відсутні, що свідчить про його чітку лінійну відокремленість.

- Сорт Iris versicolor:

  • Модель правильно класифікувала 8 зразків versicolor.

• 10 зразків, які насправді належали до versicolor, модель помилково віднесла до virginica.

• Це найбільша група помилок, що вказує на схожість ознак між цими двома сортами.

- Сорт Iris virginica:

  • Модель правильно розпізнала 10 зразків virginica.

  • 1 зразок, який насправді був virginica, був помилково класифікований як versicolor.

Загальні результати:

- Загальна кількість зразків у тестовій вибірці: 45.
- Кількість правильних прогнозів: 34 (16 + 8 + 10).
- Кількість помилкових класифікацій: 11.
- Загальна точність, що узгоджується з розрахованим показником Accuracy, становить ≈75.6%.

```
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

      setosa       1.00      1.00      1.00        16
  versicolor       0.89      0.44      0.59        18
   virginica       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.2.17. Результат

Налаштування класифікатора Ridge та їх значення

У програмі використано лінійний класифікатор Ridge (модель RidgeClassifier з бібліотеки scikit-learn), який є варіантом лінійної регресії з L2-регуляризацією. Регуляризація запобігає перенавчанню, зменшуючи вплив окремих ознак із великими вагами.

Для навчання класифікатора використано такі параметри:

- **tol = 1e-2** — поріг точності зупинки ітераційного процесу оптимізації. Коли зміна функції втрат між ітераціями стає меншою за це значення, алгоритм припиняє навчання.
- **solver = "sag"** — метод оптимізації *Stochastic Average Gradient Descent*, який ефективний для великих наборів даних і забезпечує швидку збіжність.
- **random_state = 0 (за замовчуванням)** — фіксує генератор випадкових чисел для відтворюваності результатів.

Таким чином, модель навчається лінійно відокремлювати класи, мінімізуючи суму квадратів помилок із урахуванням регуляризаційного члена, що контролює складність моделі.

Показники якості класифікації та їх результати

Для оцінки якості класифікації використано такі метрики:

- **Accuracy (Точність класифікації)** – частка правильно класифікованих прикладів від загальної кількості.→ Отримано **0.7556 (75.56%)**.
- **Precision (Точність позитивних передбачень)** – частка правильно передбачених зразків певного класу серед усіх, які модель віднесла до цього класу. → Отримано **0.8333**, що свідчить про загалом високу надійність передбачень.
- **Recall (Повнота)** – частка правильно розпізнаних зразків певного класу серед усіх реальних зразків цього класу. → Отримано **0.7556**, що вказує на добру, але не ідеальну здатність моделі знаходити всі приклади.
- **F1 Score** – гармонійне середнє між Precision та Recall, яке балансує їх вплив. → Отримано **0.7503**, що показує помірно високу якість моделі.

- **Cohen's Kappa Score** – метрика узгодженості між прогнозами моделі та реальними класами, що враховує випадкову точність. → Отримано **0.6431**, що свідчить про добру, але не ідеальну відповідність.
- **Matthews Correlation Coefficient (MCC)** – коефіцієнт кореляції між прогнозованими та істинними класами. Значення близьке до 1 вказує на високу узгодженість. → Отримано **0.6831**, що підтверджує узгодженість передбачень для всіх класів.

Коефіцієнт Коена Каппа

Це статистичний показник, який вимірює рівень узгодженості між фактичними класами та прогнозами моделі, з урахуванням ймовірності випадкового збігу.

Значення коефіцієнта змінюється від –1 до 1:

- 1 — повна узгодженість між прогнозами та реальністю;
- 0 — рівень випадкової класифікації (немає реальної узгодженості);
- <0 — гірше, ніж випадкове передбачення.

У даній роботі він= 0.6431.

Коефіцієнт кореляції Метьюза

Це показник, який оцінює збалансованість класифікації між усіма класами.

Він враховує істинно позитивні, істинно негативні, хибно позитивні та хибно негативні значення, тому надає об'єктивнішу оцінку, навіть якщо класи розподілені нерівномірно.

Значення MCC також лежить у межах від –1 до 1:

- 1 — ідеальна класифікація;
- 0 — випадкові результати;
- –1 — повна невідповідність.

Отримане значення = 0.6831

Висновок: Я використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив різні методи класифікації даних та навчився їх порівнювати

Посилання на Git: https://github.com/ipz223-zdo/AIS_Labs