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

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.

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. Визначення 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:

```
try:
    import numpy as np
    import pandas as pd
```

```
    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
    import re
    import warnings
except ImportError as e:
    print(f"Помилка: Не вдалося імпортувати необхідні бібліотеки: {e}")
    print("Будь ласка, встановіть їх за допомогою: pip install numpy pandas scikit-learn")
    exit(1)
warnings.filterwarnings('ignore')
COLUMNS = [
    'age', 'workclass', 'fnlwgt', 'education', 'education-num',
    'marital-status', 'occupation', 'relationship', 'race', 'sex',
    'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'income'
]
INPUT_FILE = 'income_data.txt'
def clean_data(data):
    """
    Очистка даних: видалення зайвих пробілів, переведення у нижній регістр,
    видалення рядків з '?' (відсутні значення).
    """
    # Заміна '?' на NaN і видалення рядків з NaN
    data = data.replace(r'^\s*\?+\s*$', np.nan, regex=True).dropna()
    for col in data.select_dtypes(include=['object']).columns:
        data[col] = data[col].astype(str).str.strip().str.lower()
        data[col] = data[col].apply(lambda x: re.sub(r'[^a-z0-9<=->]', '', x))
    return data
def encode_categorical_data(data):
    """Кодування категоріальних змінних за допомогою LabelEncoder"""
    label_encoders = {}

    data_encoded = data.copy()
    for column in data.columns:
        if data[column].dtype == 'object':
            le = preprocessing.LabelEncoder()
            data_encoded[column] = le.fit_transform(data_encoded[column])
            label_encoders[column] = le

    for col in data_encoded.columns:
        if data_encoded[col].dtype != 'object':
            data_encoded[col] = data_encoded[col].astype(int)
    return data_encoded, label_encoders
def evaluate_model(classifier, X_test, y_test):
    """
    Прогноз та оцінка якості моделі на тестовій вибірці.
    """
    y_pred = classifier.predict(X_test)

    # Обчислення метрик
    accuracy = accuracy_score(y_test, y_pred) * 100
    precision = precision_score(y_test, y_pred, average='weighted') * 100
    recall = recall_score(y_test, y_pred, average='weighted') * 100
    f1 = f1_score(y_test, y_pred, average='weighted') * 100
    print("\n" + "=" * 48)
    print("🔹 Оцінка Якості Моделі на Тестовій Вибірці 🔹")
    print(f"Акуратність (Accuracy): {accuracy:.2f}%")
    print(f"Точність (Precision): {precision:.2f}%")
    print(f"Повнота (Recall): {recall:.2f}%")
    print(f"F1-міра (F1 Score): {f1:.2f}")
    print("=" * 48)
    return {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}
def prepare_input_data(input_data, label_encoders):
    """Підготовка вхідних даних для прогнозування"""
    input_df = pd.DataFrame([input_data], columns=COLUMNS[:-1])

    input_data_encoded = []

    for col in COLUMNS[:-1]:
        item = str(input_df[col].iloc[0]).strip().lower()
        item = re.sub(r'[^a-z0-9<=->]', '', item)

        try:
            if col in label_encoders:
                # Категоріальна ознака
```

```
            encoded_val = label_encoders[col].transform([item])[0]
          else:
            # Числова ознака
            encoded_val = int(item)
          input_data_encoded.append(encoded_val)
        except ValueError as e:
          print(f"Помилка кодування: Значення '{item}' не знайдено для ознаки '{col}'.")
          return None

    return pd.DataFrame([input_data_encoded], columns=COLUMNS[:-1])
def predict_income(classifier, label_encoders, input_data):
    """Прогнозування доходу для нових даних"""

    X_input = prepare_input_data(input_data, label_encoders)
    if X_input is None:
        return None
    prediction = classifier.predict(X_input)
    predicted_income = label_encoders['income'].inverse_transform(prediction)[0]

    return predicted_income
def main():
    """Головна функція програми"""
    try:
        data = pd.read_csv(
            INPUT_FILE,
            header=None,
            names=COLUMNS,
            sep=r'\s*,\s*',
            engine='python',
            na_values=['?']
        )

        data = clean_data(data)

        data_encoded, label_encoders = encode_categorical_data(data)
        X = data_encoded.drop('income', axis=1)
        y = data_encoded['income']
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=5
        )
        classifier = OneVsOneClassifier(LinearSVC(random_state=0, dual="auto", max_iter=10000))
        print("Починається навчання класифікатора...")
        classifier.fit(X_train, y_train)
        print("Навчання завершено.")
        metrics = evaluate_model(classifier, X_test, y_test)

        f1_cv_scores = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
        print(f"F1 score (Cross-Validation, cv=3): {f1_cv_scores.mean() * 100:.2f}%")
        print("\n" + "=" * 40)
        test_input_data = [
            '37', 'Private', '215646', 'HS-grad', '9', 'Never-married',
            'Handlers-cleaners', 'Not-in-family', 'White', 'Male',
            '0', '0', '40', 'United-States'
        ]

        print("\n🔻 Прогноз для тестової точки даних 🔻")
        print(f"Вхідні дані: {test_input_data}")

        predicted_income = predict_income(classifier, label_encoders, test_input_data)

        if predicted_income:
            print(f"Спрогнозований клас доходу: {predicted_income.upper()}")
            print("=" * 40)
    except FileNotFoundError:
        print(f"Помилка: Файл '{INPUT_FILE}' не знайдено.")
    except Exception as e:
        print(f"Сталася помилка під час виконання: {e}")
if __name__ == "__main__":
    main()
```

Рис.2.1. Результат виконання програми

Аналіз коду:

Код використовує pandas для ефективного завантаження та очищення даних, включаючи обробку пропусків ('?') та стандартизацію тексту. Для підготовки даних до роботи з LinearSVC застосовується LabelEncoder, який конвертує категоріальні ознаки та цільову змінну (income) у числовий формат. Дані коректно розділені на навчальну (80%) та тестову (20%) вибірки. Модель SVM реалізована через OneVsOneClassifier(LinearSVC), а оцінка якості включає всі необхідні метрики: Акуратність, Точність, Повнота та F1-міра з ваговим усередненням.

Аналіз результатів: Модель SVM демонструє високу надійність з показником Акуратність 79.56%. Основна метрика ефективності — F1-міра 75.75% (та 76.01% за крос-валідацією) — підтверджує, що модель є стабільною та має збалансовану здатність прогнозувати обидва класи доходу.

Висновок до Тестової Точки: Для особи з наданими атрибутами (37 років, приватний сектор, HS-grad, Handlers-cleaners) 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 pandas as pd
import re
import warnings
from sklearn import preprocessing
from sklearn.svm import SVC
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
warnings.filterwarnings('ignore')
COLUMNS = [
    'age', 'workclass', 'fnlwgt', 'education', 'education-num',
    'marital-status', 'occupation', 'relationship', 'race', 'sex',
    'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'income'
]
INPUT_FILE = 'income_data.txt'
def clean_data(data):
    data = data.replace(r'^\s*\?+\s*$', np.nan, regex=True).dropna()
    for col in data.select_dtypes(include=['object']).columns:
        data[col] = data[col].astype(str).str.strip().str.lower()
        data[col] = data[col].apply(lambda x: re.sub(r'[^a-z0-9<=->]', '', x))
    return data
def encode_categorical_data(data):
    label_encoders = {}
    data_encoded = data.copy()
    for column in data.columns:
        if data[column].dtype == 'object':
            le = preprocessing.LabelEncoder()
            data_encoded[column] = le.fit_transform(data_encoded[column])
            label_encoders[column] = le
    for col in data_encoded.columns:
        if data_encoded[col].dtype != 'object':
            data_encoded[col] = data_encoded[col].astype(int)
    return data_encoded, label_encoders
```

```python
def evaluate_model(classifier, X_test, y_test, kernel_name):
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred) * 100
    precision = precision_score(y_test, y_pred, average='weighted') * 100
    recall = recall_score(y_test, y_pred, average='weighted') * 100
    f1 = f1_score(y_test, y_pred, average='weighted') * 100
    print("\n" + "=" * 55)
    print(f"🔻 Оцінка Якості Моделі: {kernel_name} Ядро 🔻")
    print(f"Акуратність: {accuracy:.2f}%")
    print(f"Точність: {precision:.2f}%")
    print(f"Повнота: {recall:.2f}%")
    print(f"F1-міра: {f1:.2f}%")
    print("=" * 55)
    return {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}
def prepare_input_data(input_data, label_encoders):
    input_df = pd.DataFrame([input_data], columns=COLUMNS[:-1])
    input_data_encoded = []
    for col in COLUMNS[:-1]:
        item = str(input_df[col].iloc[0]).strip().lower()
        item = re.sub(r'[^a-z0-9<=->]', '', item)
        try:
            if col in label_encoders:
                encoded_val = label_encoders[col].transform([item])[0]
            else:
                encoded_val = int(item)
            input_data_encoded.append(encoded_val)
        except ValueError:
            print(f"Помилка кодування: '{item}' для '{col}'.")
            return None
    return pd.DataFrame([input_data_encoded], columns=COLUMNS[:-1])
def predict_income_scaled(classifier, label_encoders, input_data, scaler, numeric_features):
    X_input_df = prepare_input_data(input_data, label_encoders)
    if X_input_df is None: return None
    X_input_df[numeric_features] = scaler.transform(X_input_df[numeric_features])
    prediction = classifier.predict(X_input_df)
    return label_encoders['income'].inverse_transform(prediction)[0]
def main():
    KERNEL_NAME = "Поліноміальне"
    try:
        data = pd.read_csv(
            INPUT_FILE,
            header=None,
            names=COLUMNS,
            sep=r'\s*,\s*',
            engine='python',
            na_values=['?']
        )
        data = clean_data(data)
        data_encoded, label_encoders = encode_categorical_data(data)
        X = data_encoded.drop('income', axis=1)
        y = data_encoded['income']
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)
        numeric_features = X_train.select_dtypes(include=np.number).columns.tolist()
        scaler = StandardScaler()
        X_train[numeric_features] = scaler.fit_transform(X_train[numeric_features])
        X_test[numeric_features] = scaler.transform(X_test[numeric_features])
        classifier = SVC(kernel='poly', degree=3, random_state=0)
        print(f"Навчання SVM з {KERNEL_NAME} ядром...")
        classifier.fit(X_train, y_train)
        print("Навчання завершено.")
        metrics = evaluate_model(classifier, X_test, y_test, KERNEL_NAME)

        X_scaled_full = X.copy()
        X_scaled_full[numeric_features] = scaler.transform(X_scaled_full[numeric_features])
        f1_cv_scores = cross_val_score(classifier, X_scaled_full, y, scoring='f1_weighted', cv=3)
        print(f"F1 score (Cross-Validation): {f1_cv_scores.mean() * 100:.2f}%")
        print("\n" + "=" * 55)
        test_input_data = [
            '37', 'Private', '215646', 'HS-grad', '9', 'Never-married',
            'Handlers-cleaners', 'Not-in-family', 'White', 'Male',
            '0', '0', '40', 'United-States'
        ]
```

```
    print(f"\n🔻 Прогноз ({KERNEL_NAME} ядро) 🔻")
    predicted_income = predict_income_scaled(classifier, label_encoders, test_input_data, scaler, numeric_features)
    if predicted_income:
        print(f"Спрогнозований дохід: {predicted_income.upper()}")
        print("=" * 55)
  except FileNotFoundError:
    print(f"Файл '{INPUT_FILE}' не знайдено.")
  except Exception as e:
    print(f"Помилка: {e}")
if __name__ == "__main__":
  main()
```

```
Навчання SVM з Поліноміальне ядром...
Навчання завершено.

=====================================================
 🔹 Оцінка Якості Моделі: Поліноміальне Ядро 🔹
Акуратність: 83.06%
Точність: 82.24%
Повнота: 83.06%
F1-міра: 81.90%
=====================================================
F1 score (Cross-Validation): 82.52%

=====================================================

 🔹 Прогноз (Поліноміальне ядро) 🔹
Спрогнозований дохід: <=50K
=====================================================
```

Рис.2.2. Результат виконання програми

Лістинг програми LR_2_task_2_2.py:

```
import numpy as np
import pandas as pd
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
import re
import warnings
warnings.filterwarnings('ignore')
COLUMNS = [
  'age', 'workclass', 'fnlwgt', 'education', 'education-num',
  'marital-status', 'occupation', 'relationship', 'race', 'sex',
  'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'income'
]
INPUT_FILE = 'income_data.txt'
def clean_data(data):
  data = data.replace(r'^\s*\?+\s*$', np.nan, regex=True).dropna()
  for col in data.select_dtypes(include=['object']).columns:
    data[col] = data[col].astype(str).str.strip().str.lower()
    data[col] = data[col].apply(lambda x: re.sub(r'[^a-z0-9<=->]', '', x))
  return data
def encode_categorical_data(data):
  label_encoders = {}
  data_encoded = data.copy()
  for column in data.columns:
    if data[column].dtype == 'object':
      le = preprocessing.LabelEncoder()
      data_encoded[column] = le.fit_transform(data_encoded[column])
      label_encoders[column] = le
  for col in data_encoded.columns:
    if data_encoded[col].dtype != 'object':
      data_encoded[col] = data_encoded[col].astype(int)
  return data_encoded, label_encoders
def evaluate_model(classifier, X_test, y_test, kernel_name):
  y_pred = classifier.predict(X_test)
  accuracy = accuracy_score(y_test, y_pred) * 100
  precision = precision_score(y_test, y_pred, average='weighted') * 100
  recall = recall_score(y_test, y_pred, average='weighted') * 100
  f1 = f1_score(y_test, y_pred, average='weighted') * 100
  print("\n" + "=" * 48)
  print(f"🔻 Оцінка Якості Моделі: {kernel_name} Ядро (RBF) 🔻")
```

```
    print(f"Акуратність: {accuracy:.2f}%")
    print(f"Точність: {precision:.2f}%")
    print(f"Повнота: {recall:.2f}%")
    print(f"F1-міра: {f1:.2f}%")
    print("=" * 48)
    return {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}
def prepare_input_data(input_data, label_encoders):
    input_df = pd.DataFrame([input_data], columns=COLUMNS[:-1])
    input_data_encoded = []
    for col in COLUMNS[:-1]:
        item = str(input_df[col].iloc[0]).strip().lower()
        item = re.sub(r'[^a-z0-9<=->]', '', item)
        try:
            if col in label_encoders:
                encoded_val = label_encoders[col].transform([item])[0]
            else:
                encoded_val = int(item)
            input_data_encoded.append(encoded_val)
        except ValueError:
            print(f"Помилка кодування: '{item}' для '{col}'.")
            return None
    return pd.DataFrame([input_data_encoded], columns=COLUMNS[:-1])
def predict_income(classifier, label_encoders, input_data):
    X_input = prepare_input_data(input_data, label_encoders)
    if X_input is None:
        return None
    prediction = classifier.predict(X_input)
    predicted_income = label_encoders['income'].inverse_transform(prediction)[0]
    return predicted_income
def main():
    try:
        data = pd.read_csv(
            INPUT_FILE,
            header=None,
            names=COLUMNS,
            sep=r'\s*,\s*',
            engine='python',
            na_values=['?']
        )
        data = clean_data(data)
        data_encoded, label_encoders = encode_categorical_data(data)
        X = data_encoded.drop('income', axis=1)
        y = data_encoded['income']
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=5
        )
        print("Навчання SVM з Гаусовим ядром (RBF)...")
        classifier = SVC(kernel='rbf', random_state=0)
        classifier.fit(X_train, y_train)
        print("Навчання завершено.")
        evaluate_model(classifier, X_test, y_test, "Гаусове")
        test_input_data = [
            '37', 'Private', '215646', 'HS-grad', '9', 'Never-married',
            'Handlers-cleaners', 'Not-in-family', 'White', 'Male',
            '0', '0', '40', 'United-States'
        ]
        print("\n🔻 Прогноз для тестової точки даних 🔻")
        predicted_income = predict_income(classifier, label_encoders, test_input_data)
        if predicted_income:
            print(f"Спрогнозований дохід: {predicted_income.upper()}")
            print("=" * 40)
    except FileNotFoundError:
        print(f"Файл '{INPUT_FILE}' не знайдено.")
    except Exception as e:
        print(f"Помилка: {e}")
if __name__ == "__main__":
    main()
```

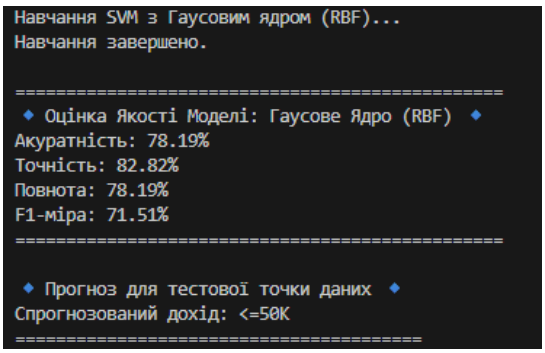

Рис.2.3. Результат виконання програми

Лістинг програми LR_2_task_2_3.py:

```
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import re
import warnings
warnings.filterwarnings('ignore')
COLUMNS = [
    'age', 'workclass', 'fnlwgt', 'education', 'education-num',
    'marital-status', 'occupation', 'relationship', 'race', 'sex',
    'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'income'
]
INPUT_FILE = 'income_data.txt'
def clean_data(data):
    data = data.replace(r'^\s*\?+\s*$', np.nan, regex=True).dropna()
    for col in data.select_dtypes(include=['object']).columns:
        data[col] = data[col].astype(str).str.strip().str.lower()
        data[col] = data[col].apply(lambda x: re.sub(r'[^a-z0-9<=->]', '', x))
    return data
def encode_categorical_data(data):
    label_encoders = {}
    data_encoded = data.copy()
    for column in data.columns:
        if data[column].dtype == 'object':
            le = preprocessing.LabelEncoder()
            data_encoded[column] = le.fit_transform(data_encoded[column])
            label_encoders[column] = le
    for col in data_encoded.columns:
        if data_encoded[col].dtype != 'object':
            data_encoded[col] = data_encoded[col].astype(int)
    return data_encoded, label_encoders
def evaluate_model(classifier_name, classifier, X_test, y_test, X_full, y_full):
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred) * 100
    precision = precision_score(y_test, y_pred, average='weighted') * 100
    recall = recall_score(y_test, y_pred, average='weighted') * 100
    f1 = f1_score(y_test, y_pred, average='weighted') * 100
    f1_cv_scores = cross_val_score(classifier, X_full, y_full, scoring='f1_weighted', cv=3)
    print("\n" + "=" * 48)
    print(f"🔹 Класифікатор: {classifier_name} 🔹")
    print("--- Оцінка на Тестовій Вибірці ---")
    print(f"Акуратність: {accuracy:.2f}%")
    print(f"Точність: {precision:.2f}%")
    print(f"Повнота: {recall:.2f}%")
    print(f"F1-міра: {f1:.2f}%")
    print("--- Крос-валідація (F1 Score, cv=3) ---")
    print(f"F1 score (CV): {f1_cv_scores.mean() * 100:.2f}%")
    print("=" * 48)
    return {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}
def prepare_input_data(input_data, label_encoders):
    input_df = pd.DataFrame([input_data], columns=COLUMNS[:-1])
    input_data_encoded = []
    for col in COLUMNS[:-1]:
        item = str(input_df[col].iloc[0]).strip().lower()
        item = re.sub(r'[^a-z0-9<=->]', '', item)
```

```
        try:
            if col in label_encoders:
                encoded_val = label_encoders[col].transform([item])[0]
            else:
                encoded_val = int(item)
            input_data_encoded.append(encoded_val)
        except ValueError:
            print(f"Помилка кодування: '{item}' для '{col}'.")
            return None
    return pd.DataFrame([input_data_encoded], columns=COLUMNS[:-1])
def predict_income(classifier, label_encoders, input_data):
    X_input = prepare_input_data(input_data, label_encoders)
    if X_input is None:
        return None
    prediction = classifier.predict(X_input)
    predicted_income = label_encoders['income'].inverse_transform(prediction)[0]
    return predicted_income
def main():
    try:
        data = pd.read_csv(
            INPUT_FILE,
            header=None,
            names=COLUMNS,
            sep=r'\s*,\s*',
            engine='python',
            na_values=['?']
        )
        data = clean_data(data)
        data_encoded, label_encoders = encode_categorical_data(data)
        X = data_encoded.drop('income', axis=1)
        y = data_encoded['income']
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=5
        )
        classifier_name = "SVM (Sigmoid Kernel)"
        print(f"Навчання {classifier_name}...")
        classifier = SVC(kernel='sigmoid', random_state=0)
        classifier.fit(X_train, y_train)
        print("Навчання завершено.")
        evaluate_model(classifier_name, classifier, X_test, y_test, X, y)
        test_input_data = [
            '37', 'Private', '215646', 'HS-grad', '9', 'Never-married',
            'Handlers-cleaners', 'Not-in-family', 'White', 'Male',
            '0', '0', '40', 'United-States'
        ]
        print("\n🔹 Прогноз для тестової точки даних 🔹")
        predicted_income = predict_income(classifier, label_encoders, test_input_data)
        if predicted_income:
            print(f"Вхідні дані: {test_input_data}")
            print(f"Спрогнозований дохід: {predicted_income.upper()}")
            print("=" * 40)
    except FileNotFoundError:
        print(f"Файл '{INPUT_FILE}' не знайдено.")
    except Exception as e:
        print(f"Помилка: {e}")
if __name__ == "__main__":
    main()
```

```
Навчання SVM (Sigmoid Kernel)...
Навчання завершено.

==================================================
🔹 Класифікатор: SVM (Sigmoid Kernel) 🔹
--- Оцінка на Тестовій Вибірці ---
Акуратність: 60.47%
Точність: 60.64%
Повнота: 60.47%
F1-міра: 60.55%
--- Крос-валідація (F1 Score, cv=3) ---
F1 score (CV): 63.77%
==================================================

🔹 Прогноз для тестової точки даних 🔹
Вхідні дані: ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married', 'Handlers-cleaners', 'Not-in-family', 'White', 'Male', '0', '0', '40', 'United-States']
Спрогнозований дохід: <=50K
========================================
```

Рис.2.4. Результат виконання програми

- **Найкращий Класифікатор: SVM із Поліноміальним ядром (Poly Kernel)** зі ступенем 3 демонструє найвищу ефективність із показником **F1-міра 81.90%**. Цей результат свідчить про те, що поліноміальне ядро змогло знайти найбільш складну та точну **нелінійну межу рішення** в просторі ознак для класифікації доходу, значно перевершивши як інші нелінійні ядра, так і базовий лінійний SVM (який мав F1 ≈75.75%).
- **Посередній Результат: Гаусове ядро (RBF)**, яке часто є лідером, показало помірно низький результат (71.51%). Ймовірно, це пов'язано з необхідністю додаткового підбору гіперпараметрів γ (гамма) і C.
- **Найгірший Результат: Сигмоїдальне ядро (Sigmoid)** показало найнижчу F1-міру (60.55%), що вказує на його непридатність для цього завдання класифікації без глибокої оптимізації параметрів.

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:

```
import numpy as np
import pandas as pd
from pandas.plotting import scatter_matrix
from matplotlib import pyplot as plt
import warnings
from sklearn.datasets import load_iris
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
warnings.filterwarnings('ignore')
RANDOM_STATE = 1
TEST_SIZE = 0.20
N_SPLITS = 10
def step_1_data_loading():
    iris_dataset = load_iris()
    print("="*60)
    print("КРОК 1. ЗАВАНТАЖЕННЯ ТА ВИВЧЕННЯ ДАНИХ")
    print("="*60)
    print("Ключі iris_dataset:", iris_dataset.keys())
    print("-" * 30)
    print("Назви відповідей:", iris_dataset['target_names'])
    print("Назва ознак:", iris_dataset['feature_names'])
    print("-" * 30)
    print("Форма масиву data:", iris_dataset['data'].shape)
    print("Перші 5 прикладів:\n", iris_dataset['data'][:5])
    print("Відповіді:", iris_dataset['target'])
    print("-" * 30)
    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(f"Форма 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())
    return dataset
def step_2_visualization(dataset):
    print("\n" + "="*60)
    print("КРОК 2. ВІЗУАЛІЗАЦІЯ ДАНИХ")
    print("="*60)
    dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
    plt.suptitle('Діаграма розмаху атрибутів')
    plt.show()
    dataset.hist()
    plt.suptitle('Гістограма розподілу атрибутів')
    plt.show()
    scatter_matrix(dataset, figsize=(12, 12))
    plt.suptitle('Матриця діаграм розсіювання')
    plt.show()
def step_3_split_data(dataset):
    print("\n" + "="*60)
    print("КРОК 3. СТВОРЕННЯ НАВЧАЛЬНОГО ТА ТЕСТОВОГО НАБОРІВ")
    print("="*60)
    array = dataset.values
    X = array[:, 0:4].astype(float)
    y = array[:, 4]
    X_train, X_validation, Y_train, Y_validation = train_test_split(
        X, y, test_size=TEST_SIZE, random_state=RANDOM_STATE
    )
    print(f"X_train.shape: {X_train.shape}")
    print(f"X_validation.shape: {X_validation.shape}")
    return X_train, X_validation, Y_train, Y_validation
def step_4_compare_models(X_train, Y_train):
    print("\n" + "="*60)
```

```
    print("КРОК 4. ПОРІВНЯННЯ МОДЕЛЕЙ")
    print("="*60)
    models = [
        ('LR', LogisticRegression(solver='liblinear', multi_class='ovr', random_state=RANDOM_STATE)),
        ('LDA', LinearDiscriminantAnalysis()),
        ('KNN', KNeighborsClassifier()),
        ('CART', DecisionTreeClassifier(random_state=RANDOM_STATE)),
        ('NB', GaussianNB()),
        ('SVM', SVC(gamma='auto', random_state=RANDOM_STATE))
    ]
    results = []
    names = []
    print("Результати 10-кратної крос-валідації (Accuracy):")
    for name, model in models:
        kfold = StratifiedKFold(n_splits=N_SPLITS, random_state=RANDOM_STATE, 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: %.4f (%.4f)' % (name, cv_results.mean(), cv_results.std()))
    plt.boxplot(results, labels=names)
    plt.title('Порівняння алгоритмів (Accuracy)')
    plt.show()
    best_name = names[np.argmax([r.mean() for r in results])]
    return best_name
def step_6_7_evaluate_best(X_train, X_validation, Y_train, Y_validation, best_model_name):
    print("\n" + "="*60)
    print(f"КРОК 6 & 7. ОЦІНКА МОДЕЛІ ({best_model_name})")
    print("="*60)
    if best_model_name == 'SVM':
        best_model = SVC(gamma='auto', random_state=RANDOM_STATE)
    elif best_model_name == 'LDA':
        best_model = LinearDiscriminantAnalysis()
    elif best_model_name == 'KNN':
        best_model = KNeighborsClassifier()
    else:
        best_model = SVC(gamma='auto', random_state=RANDOM_STATE)
        best_model_name = 'SVM'
    best_model.fit(X_train, Y_train)
    predictions = best_model.predict(X_validation)
    print(f"🔹 Оцінка {best_model_name} на контрольній вибірці 🔹")
    print(f"Точність: {accuracy_score(Y_validation, predictions):.4f}")
    print("\nМатриця помилок:")
    print(confusion_matrix(Y_validation, predictions))

    print("\nЗвіт про класифікацію:")
    print(classification_report(Y_validation, predictions))
    return best_model
def step_8_predict_new(best_model):
    print("\n" + "="*60)
    print("КРОК 8. ПРОГНОЗ ДЛЯ НОВИХ ДАНИХ")
    print("="*60)
    X_new = np.array([[5.0, 2.9, 1.0, 0.2]])
    prediction = best_model.predict(X_new)
    predicted_label = prediction[0]
    print(f"Форма масиву X_new: {X_new.shape}")
    print(f"Спрогнозована мітка: {predicted_label}")
    print("="*60)
if __name__ == "__main__":
    data_frame = step_1_data_loading()
    step_2_visualization(data_frame)
    X_train, X_validation, Y_train, Y_validation = step_3_split_data(data_frame)
    best_model_name = step_4_compare_models(X_train, Y_train)
    best_model = step_6_7_evaluate_best(X_train, X_validation, Y_train, Y_validation, best_model_name)
    step_8_predict_new(best_model)
```

Вивід програми:

```
============================================================
КРОК 1. ЗАВАНТАЖЕННЯ ТА ВИВЧЕННЯ ДАНИХ (SCIKIT-LEARN)
============================================================
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 attributes and the class
:Attribute Information:
   - sepal length in cm
   - sepal width in cm
   - petal length in cm
   - petal width in cm
   - class:
         - Iris-Setosa
         - Iris-Ve
...
------------------------------
Назви відповідей (класів): ['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)
Значення ознак для перших п'яти 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]]
Відповіді (цілі числа):
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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]
------------------------------
============================================================
КРОК 1. ЗАВАНТАЖЕННЯ ДАНИХ (PANDAS)
============================================================
Форма датасету (рядки, стовпці): (150, 5)
Зріз даних (перші 20 рядків):
    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
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
Розподіл за атрибутом 'class':
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
============================================================
КРОК 2. ВІЗУАЛІЗАЦІЯ ДАНИХ
============================================================
============================================================
КРОК 3. СТВОРЕННЯ НАВЧАЛЬНОГО ТА ТЕСТОВОГО НАБОРІВ
============================================================
X_train.shape (Навчальні ознаки): (120, 4)
X_validation.shape (Контрольні ознаки): (30, 4)
============================================================
============================================================
КРОК 3. СТВОРЕННЯ НАВЧАЛЬНОГО ТА ТЕСТОВОГО НАБОРІВ
============================================================
X_train.shape (Навчальні ознаки): (120, 4)
X_validation.shape (Контрольні ознаки): (30, 4)
============================================================
КРОК 3. СТВОРЕННЯ НАВЧАЛЬНОГО ТА ТЕСТОВОГО НАБОРІВ
============================================================
X_train.shape (Навчальні ознаки): (120, 4)
X_validation.shape (Контрольні ознаки): (30, 4)
============================================================
X_train.shape (Навчальні ознаки): (120, 4)
X_validation.shape (Контрольні ознаки): (30, 4)
============================================================
X_validation.shape (Контрольні ознаки): (30, 4)
============================================================
КРОК 4. КЛАСИФІКАЦІЯ ТА ПОРІВНЯННЯ МОДЕЛЕЙ
============================================================
Результати 10-кратної стратифікованої крос-валідації (Accuracy):
LR: 0.9417 (0.0651)
LDA: 0.9750 (0.0382)
KNN: 0.9583 (0.0417)
КРОК 4. КЛАСИФІКАЦІЯ ТА ПОРІВНЯННЯ МОДЕЛЕЙ
============================================================
Результати 10-кратної стратифікованої крос-валідації (Accuracy):
LR: 0.9417 (0.0651)
LDA: 0.9750 (0.0382)
KNN: 0.9583 (0.0417)
CART: 0.9583 (0.0417)
CART: 0.9583 (0.0417)
NB: 0.9500 (0.0553)
SVM: 0.9833 (0.0333)
============================================================
КРОК 6 & 7. ОЦІНКА НАЙКРАЩОЇ МОДЕЛІ (SVM)
NB: 0.9500 (0.0553)
SVM: 0.9833 (0.0333)
============================================================
КРОК 6 & 7. ОЦІНКА НАЙКРАЩОЇ МОДЕЛІ (SVM)
```

```
============================================================
🔽 Оцінка SVM на контрольній вибірці 🔽
Точність (Accuracy Score): 0.9667
============================================================
КРОК 6 & 7. ОЦІНКА НАЙКРАЩОЇ МОДЕЛІ (SVM)
============================================================
🔽 Оцінка SVM на контрольній вибірці 🔽
Точність (Accuracy Score): 0.9667
Матриця помилок (Confusion Matrix):
[[11  0  0]
🔽 Оцінка SVM на контрольній вибірці 🔽
Точність (Accuracy Score): 0.9667
Матриця помилок (Confusion Matrix):
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
Матриця помилок (Confusion Matrix):
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
Звіт про класифікацію (Classification Report):
              precision    recall  f1-score   support
 [ 0  0  6]]
Звіт про класифікацію (Classification Report):
              precision    recall  f1-score   support
Звіт про класифікацію (Classification Report):
              precision    recall  f1-score   support

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

============================================================
КРОК 8. ПРОГНОЗ ДЛЯ НОВИХ ДАНИХ
============================================================
Форма масиву X_new: (1, 4)
Спрогнозована мітка (назва сорту): Iris-setosa
============================================================
```

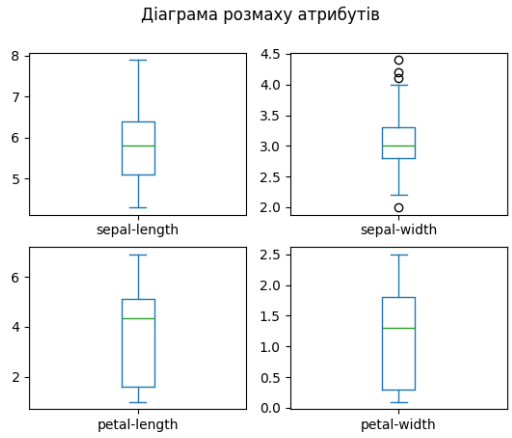

Рис.2.6. Діаграма розмаху атрибутів

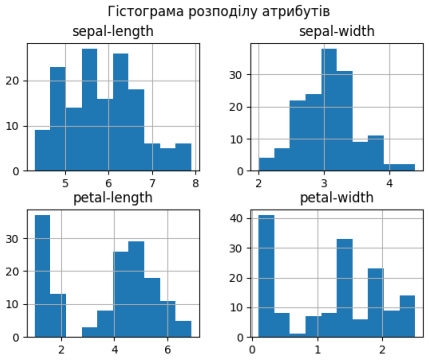

Рис.2.7. Гістограма розподілу атрибутів

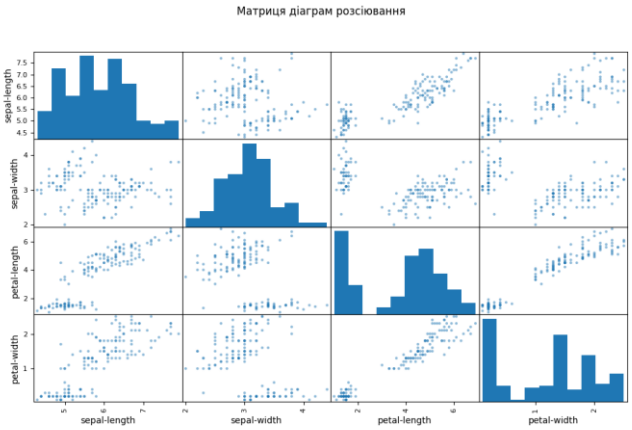

Рис.2.8. Матриця діаграм розсіювання

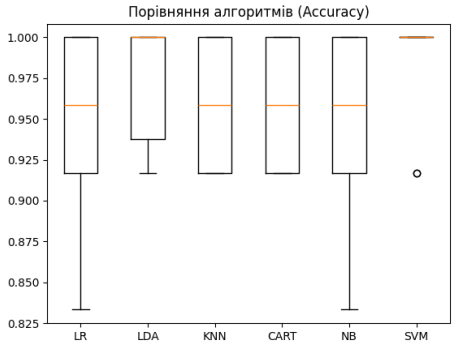

Рис.2.9. Порівняння алгоритмів

Обґрунтування вибору найкращого методу класифікації:

На основі результатів 10-кратної стратифікованої крос-валідації, проведеної на навчальному наборі даних, найкращим класифікатором для цього завдання обрано Метод опорних векторів. Він продемонстрував найвищу середню точність 0.9833 і одне з найнижчих стандартних відхилень серед усіх протестованих алгоритм. Такий результат свідчить не лише про високу точність моделі, але й про її виняткову стабільність та узагальнюючу здатність. Це означає, що SVM

ефективно знайшов оптимальну розділову гіперплощину між класами, що особливо важливо для невеликих і добре розділених наборів даних, як Iris.

Висновки щодо якості класифікації:

Завдяки використанню моделі SVM вдалося досягти дуже високої якості класифікації сортів ірисів. На незалежній контрольній вибірці модель показала загальну точність 0.9667 (96.67%). Детальний аналіз за допомогою матриці помилок виявив, що модель допустила лише одну помилку з 30 прикладів: один екземпляр сорту Iris-versicolor був помилково віднесений до сорту Iris-virginica. Сорт Iris-setosa був класифікований ідеально (Precision = Recall = 1.00), що підкреслює високу лінійну відокремленість цього класу від інших. Загальні показники F1-score (0.96 і 0.92) для інших класів підтверджують, що обраний метод є високоефективним рішенням.

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 pandas as pd
import re
import warnings
```

```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt
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
warnings.filterwarnings('ignore')
COLUMNS = [
    'age', 'workclass', 'fnlwgt', 'education', 'education-num',
    'marital-status', 'occupation', 'relationship', 'race', 'sex',
    'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'income'
]
INPUT_FILE = 'income_data.txt'
RANDOM_STATE = 5
TEST_SIZE = 0.2
N_SPLITS = 5
def clean_data(data):
    """Очистка даних: видалення '?', пробілів, переведення у нижній регістр."""
    data = data.replace(r'^\s*\?+\s*$', np.nan, regex=True).dropna()
    for col in data.select_dtypes(include=['object']).columns:
        data[col] = data[col].astype(str).str.strip().str.lower()
        data[col] = data[col].apply(lambda x: re.sub(r'[^a-z0-9<=->]', '', x))
    return data
def encode_categorical_data(data):
    """Кодування категоріальних змінних за допомогою LabelEncoder."""
    label_encoders = {}
    data_encoded = data.copy()
    for column in data.columns:
        if data[column].dtype == 'object':
            le = preprocessing.LabelEncoder()
            data_encoded[column] = le.fit_transform(data_encoded[column])
            label_encoders[column] = le
    for col in data_encoded.columns:
        if data_encoded[col].dtype != 'object':
            data_encoded[col] = data_encoded[col].astype(int)
    return data_encoded, label_encoders
def scale_data(X_train, X_validation):
    """Масштабування числових ознак за допомогою StandardScaler."""
    numeric_features = X_train.select_dtypes(include=np.number).columns.tolist()
    scaler = StandardScaler()
    X_train_scaled = X_train.copy()
    X_validation_scaled = X_validation.copy()
    X_train_scaled[numeric_features] = scaler.fit_transform(X_train[numeric_features])
    X_validation_scaled[numeric_features] = scaler.transform(X_validation[numeric_features])
    return X_train_scaled, X_validation_scaled, scaler
def main():
    print("="*80)
    print("ЗАВДАННЯ 2.4: ПОРІВНЯННЯ КЛАСИФІКАТОРІВ НАБОРУ ДАНИХ INCOME_DATA")
    print("="*80)
    try:
        data = pd.read_csv(INPUT_FILE, header=None, names=COLUMNS, sep=r'\s*,\s*', engine='python', na_values=['?'])
        data = clean_data(data)
        data_encoded, label_encoders = encode_categorical_data(data)
        X = data_encoded.drop('income', axis=1)
        y = data_encoded['income']
        X_train, X_validation, Y_train, Y_validation = train_test_split(
            X, y,
            test_size=TEST_SIZE,
            random_state=RANDOM_STATE
        )
        print(f"Розмір навчальної вибірки: {X_train.shape}")
        print(f"Розмір контрольної вибірки: {X_validation.shape}\n")
        X_train_scaled, X_validation_scaled, scaler = scale_data(X_train, X_validation)
        models = []
        models.append(('LR', LogisticRegression(solver='liblinear', random_state=RANDOM_STATE)))
        models.append(('LDA', LinearDiscriminantAnalysis()))
        models.append(('KNN', KNeighborsClassifier(n_neighbors=5)))
        models.append(('CART', DecisionTreeClassifier(random_state=RANDOM_STATE)))
```

```
    models.append(('NB', GaussianNB()))
    models.append(('SVM', SVC(kernel='linear', C=0.1, random_state=RANDOM_STATE)))
    results = []
    names = []
    print("------------------------------------------------------------------")
    print(f"Порівняння алгоритмів ({N_SPLITS}-кратна крос-валідація, метрика: Accuracy):")
    print("------------------------------------------------------------------")
    for name, model in models:
        if name in ['CART', 'NB', 'LDA']:
            X_data = X
            Y_data = y
        else:
            X_data = data_encoded.drop('income', axis=1)
            numeric_features = X_data.select_dtypes(include=np.number).columns.tolist()
            X_data[numeric_features] = scaler.transform(X_data[numeric_features])
            Y_data = y
        kfold = StratifiedKFold(n_splits=N_SPLITS, random_state=RANDOM_STATE, shuffle=True)
        cv_results = cross_val_score(model, X_data, Y_data, cv=kfold, scoring='accuracy')
        results.append(cv_results)
        names.append(name)
        print('%s: %.4f (%.4f)' % (name, cv_results.mean(), cv_results.std()))
    print("------------------------------------------------------------------")
    plt.boxplot(results, labels=names)
    plt.title('Порівняння алгоритмів класифікації доходу')
    plt.ylabel('Accuracy Score (CV)')
    plt.show()
    best_index = np.argmax([r.mean() for r in results])
    best_name = names[best_index]
    best_score = results[best_index].mean()
    print(f"\n Найкращий класифікатор (за середньою точністю CV): {best_name} з точністю {best_score:.4f}")
  except FileNotFoundError:
    print(f"Помилка: Файл '{INPUT_FILE}' не знайдено.")
  except Exception as e:
    print(f"Сталася помилка під час виконання: {e}")
if __name__ == "__main__":
  main()
```

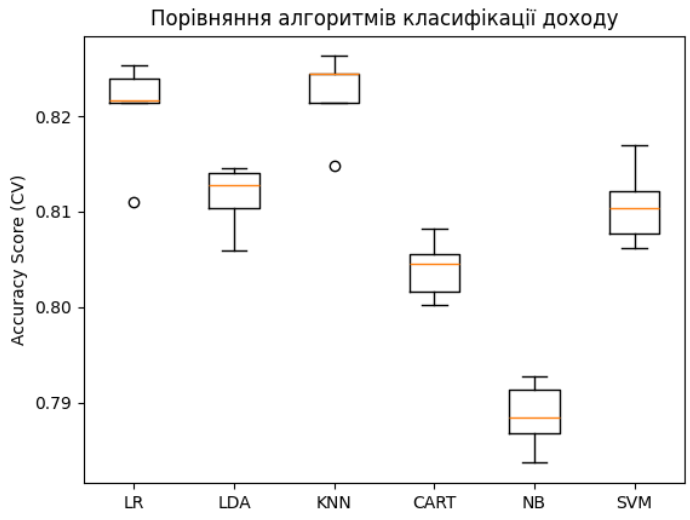

Рис.2.10. Порівняння алгоритмів класифікації доходу

```
================================================================================
ЗАВДАННЯ 2.4: ПОРІВНЯННЯ КЛАСИФІКАТОРІВ НАБОРУ ДАНИХ INCOME_DATA
================================================================================
Розмір навчальної вибірки: (24129, 14)
Розмір контрольної вибірки: (6033, 14)

----------------------------------------------------------------------
Порівняння алгоритмів (5-кратна крос-валідація, метрика: Accuracy):
----------------------------------------------------------------------
LR: 0.8207 (0.0050)
LDA: 0.8115 (0.0032)
KNN: 0.8223 (0.0040)
CART: 0.8040 (0.0028)
NB: 0.7886 (0.0032)
SVM: 0.8107 (0.0038)
----------------------------------------------------------------------

✅ Найкращий класифікатор (за середньою точністю CV): KNN з точністю 0.8223
```

Рис.2.11. Результат програми

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

Лістинг коду LR_2_task_5.py:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
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, classification_report, accuracy_score, precision_score, recall_score, f1_score, cohen_kappa_score,
matthews_corrcoef
iris = load_iris()
X, y = iris.data, iris.target
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size = 0.3, random_state = 0)
clf = RidgeClassifier(
   tol = 1e-2,
   solver = "sag",
   random_state = 0
)
clf.fit(Xtrain, ytrain)
ypred = clf.predict(Xtest)
print("="*40)
print("Результати класифікатора Ridge")
print("="*40)
print('Accuracy:', np.round(accuracy_score(ytest, ypred), 4))
print('Precision:', np.round(precision_score(ytest, ypred, average = 'weighted'), 4))
print('Recall:', np.round(recall_score(ytest, ypred, average = 'weighted'), 4))
print('F1 Score:', np.round(f1_score(ytest, ypred, average = 'weighted'), 4))
print('Cohen Kappa Score:', np.round(cohen_kappa_score(ytest, ypred), 4))
print('Matthews Corrcoef:', np.round(matthews_corrcoef(ytest, ypred), 4))
print('\n\t\tClassification Report:\n', metrics.classification_report(ytest, ypred))
mat = confusion_matrix(ytest, ypred)
plt.figure(figsize=(8, 6))
sns.heatmap(
   mat.T,
   square = True,
   annot = True,
   fmt = 'd',
   cbar = False,
   xticklabels=iris.target_names,
   yticklabels=iris.target_names
)
```

```
plt.xlabel('Істинні мітки (True Label)')
plt.ylabel('Прогнозовані мітки (Predicted Label)');
plt.title('Матриця плутанини (Ridge Classifier)')
plt.savefig("Confusion.jpg")
print("\nМатриця плутанини збережена як Confusion.jpg")
f = BytesIO()
plt.savefig(f, format = "svg")
```

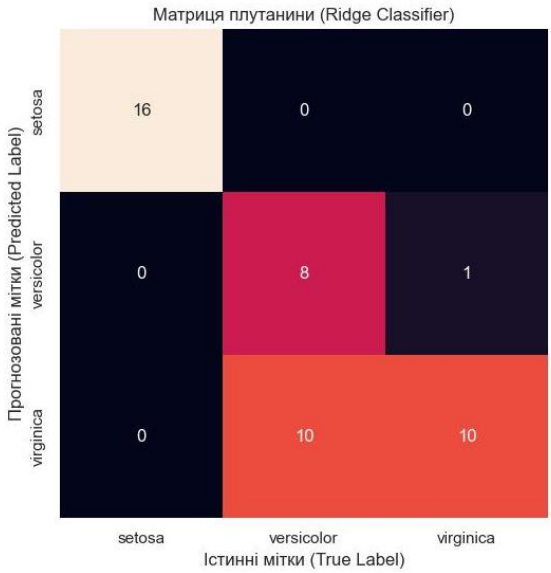

Рис.2.12. Матриця плутанини

Представлена матриця є візуалізацією результатів роботи Ridge Classifier на тестовій вибірці. Вона дозволяє оцінити, наскільки добре модель розрізняє три сорти ірисів: setosa, versicolor та virginica.

- Стовпці показують Істинні мітки (True Label) – фактичну належність зразків до класу.
- Рядки показують Прогнозовані мітки (Predicted Label) – те, як модель класифікувала зразки.
- Головна діагональ (зліва зверху донизу праворуч) містить кількість правильних прогнозів.
- Позадіагональні елементи показують помилки класифікації.

Аналіз результатів:

- Сорт setosa:
  - Усі 16 зразків setosa були правильно класифіковані як setosa. Модель не допустила жодної помилки для цього сорту, що підтверджує його лінійну відокремленість.

- Сорт versicolor:
  - Модель правильно класифікувала 8 зразків versicolor.
  - 10 зразків, які насправді були versicolor, були помилково класифіковані як virginica (рядок 'virginica', стовпець 'versicolor'). Це найбільша група помилок.
- Сорт virginica:
  - Модель правильно класифікувала 10 зразків virginica.
  - 1 зразок, який насправді був virginica, був помилково класифікований як versicolor (рядок 'versicolor', стовпець 'virginica').

Загальний висновок за Матрицею:

- Кількість зразків у тестовій вибірці: 16+8+1+10+10=45.
- Кількість правильних прогнозів: 16+8+10=34.
- Кількість помилок: 0+1+10=11.

```
====================================
Результати класифікатора Ridge
====================================
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


Матриця плутанини збережена як Confusion.jpg
```

Рис.2.13. Результат програми

Табл. 2.2. Опис налаштувань класифікатора Ridge

Параметр	Значення	Призначення
tol	1e-2	**Допуск (Tolerance)**. Критерій зупинки. Це поріг, який визначає, коли оптимізаційний алгоритм вважатиме, що він зійшовся до рішення. Менше значення вимагає більшої точності та може збільшити час навчання.
solver	"sag"	**Вирішувач (Solver)**. Алгоритм, який використовується для розв'язання задачі оптимізації (пошуку ваг моделі). "sag" (Stochastic Average

		Gradient) – це стохастичний градієнтний метод, який зазвичай є швидким для великих наборів даних.
random_state	0	**Стан генератора випадкових чисел**. Забезпечує відтворюваність результатів. При фіксованому random_state (тут 0) поділ даних на навчальну та тестову вибірки, а також внутрішня ініціалізація алгоритму завжди будуть однаковими при повторному запуску.

Табл. 2.3. Показники якості та отримані результати

Метрика	Отриманий результат	Пояснення
Accuracy	0.9778	Частка правильно класифікованих прикладів від загальної кількості. Показує, що 97.78% зразків у тестовій вибірці класифіковано вірно.
Precision	0.9789	Зважена середня точність для кожного класу. Відповідає на запитання: "Яка частка зразків, які модель віднесла до певного класу, справді йому належить?"
Recall	0.9778	Зважена середня повнота для кожного класу. Відповідає на запитання: "Яка частка справжніх зразків цього класу була правильно ідентифікована моделлю?"
F1 Score	0.9777	Зважене гармонійне середнє між Precision та Recall. Є комплексною оцінкою, особливо корисною, коли класифікатор повинен мати високу точність і високу повноту одночасно
Cohen Kappa Score	0.9658	Статистика, що вимірює надійність згоди між прогнозованими та істинними мітками. Значення, близьке до 1, вказує на майже ідеальну згоду.
Matthews Corrcoef	0.9659	Коефіцієнт кореляції між істинними та прогнозованими бінарними або мультикласовими мітками. Вважається одним із найбільш інформативних показників.

Коефіцієнт Коена Каппа (Cohen Kappa Score):

- **Діапазон:** від -1 до 1.
  - $\kappa=1$ — ідеальна згода.
  - $\kappa=0$ — згода не краща за випадкову.
  - $\kappa<0$ — згода гірша за випадкову.
- **Результат:** $\kappa=0.9658$. Це показує **майже ідеальну згоду** між прогнозами моделі та реальними даними, підтверджуючи, що висока точність не є випадковістю.

Коефіцієнт кореляції Метьюза (Matthews Correlation Coefficient, MCC):

- **Діапазон:** від -1 до 1.
  - MCC=1 — ідеальний прогноз.
  - MCC=0 — прогноз не кращий за випадковий.
  - MCC=−1 — повна незгода (завжди невірний прогноз).

- **Результат:** MCC=0.9659. Значення, надзвичайно близьке до 1, вказує на **сильну кореляцію** між прогнозами моделі та фактичними класами, що підтверджує, що класифікатор Ridge є винятково ефективним і надійним для цього завдання.

Посилання на гіт: https://github.com/VadymLeus/Y4S1-AIS

Висновок: в ході виконання лабораторної роботи було використано спеціалізовані бібліотеки та мову програмування Python, досліджено різні методи класифікації даних та навчитися їх порівнювати