ЛАБОРАТОРНА РОБОТА №2

Тема: ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ

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

Хід роботи

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

Лістинг програми

```
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.multiclass import OneVsOneClassifier

input_file = 'income_data.txt'

X = []
y = []
max_samples_per_class = 25000
class1_count, class2_count = 0, 0

with open(input_file, 'r') as file:
    for line in file:
        if '?' in line:
            continue
        data = line.strip().split(', ')
        if data[-1] == '<=50K' and class1_count < max_samples_per_class:
            X.append(data[:-1])
            y.append(0)
            class1_count += 1
        elif data[-1] == '>50K' and class2_count < max_samples_per_class:
            X.append(data[:-1])
            y.append(1)
            class2_count += 1
        if class1_count >= max_samples_per_class and class2_count >=
max_samples_per_class:
            break

X = np.array(X)
y = np.array(y)

encoders = []
X_encoded = np.empty_like(X, dtype=int)
for i in range(X.shape[1]):
    column = X[:, i]
```

```
    if not np.char.isnumeric(column).all():
        encoder = LabelEncoder()
        X_encoded[:, i] = encoder.fit_transform(column)
        encoders.append(encoder)
    else:
        X_encoded[:, i] = column.astype(int)

X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2,
random_state=42)

classifier = OneVsOneClassifier(LinearSVC(random_state=42))
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")

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

new_data_encoded = []

for i, item in enumerate(new_data):
    if item.isdigit():
        new_data_encoded.append(int(item))
    else:
        if i < len(encoders):  # Перевірка, чи є енкодер для цієї категорії
            try:
                encoded_value = encoders[i].transform([item])[0]
                new_data_encoded.append(encoded_value)
            except ValueError:  # Якщо значення не знайдено в енкодері
                encoders[i].fit([item])  # Перенавчаємо енкодер для додавання но-
вого значення
                encoded_value = encoders[i].transform([item])[0]
                new_data_encoded.append(encoded_value)
        else:
            new_data_encoded.append(-1)  # Якщо для цієї категорії немає енкодера,
присвоюємо -1

new_data_encoded = np.array(new_data_encoded)

predicted_class = classifier.predict([new_data_encoded])
predicted_income = '<=50K' if predicted_class == 0 else '>50K'

print(f"Прогнозований дохід для нового зразка: {predicted_income}")
```

Результат виконання програми:

```
"C:\Program Files\Python313\python.exe" "D:\Лаби\4 КУРС\Системи штучного інтелекту\lab2\LR_2_task_1.py"
Accuracy: 0.80
Precision: 0.80
Recall: 0.80
F1 Score: 0.76
Прогнозований дохід для нового зразка: <=50K

Process finished with exit code 0
```

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

Age — Вік особи — Тип **Числовий**
Workclass — Тип зайнятості (наприклад, приватний сектор, державна служба) — Тип **Категоріальний**
Fnlwgt — Вага, яку можна використовувати для представлення населення в наборі даних — Тип **Числовий**
Education — Рівень освіти (наприклад, бакалавр, магістр, середня освіта тощо) — Тип **Категоріальний**
Education-num — Числовий еквівалент рівня освіти (наприклад, 13 — для середньої освіти, 16 — для бакалавра) — Тип **Числовий**
Marital-status — Сімейний стан (наприклад, одружений, неодружений, вдовець) — Тип **Категоріальний**
Occupation — Рід діяльності чи професія (наприклад, інженер, медсестра, продавець) — Тип **Категоріальний**
Relationship — Статус у родині (наприклад, чоловік, дружина, дочка) — Тип **Категоріальний**
Race — Расова належність (наприклад, біла, чорна, азіатська) — Тип **Категоріальний**
Sex — Стать особи (чоловік або жінка) — Тип **Категоріальний**
Capital-gain — Капітальний прибуток (дохід від інвестицій, наприклад, від продажу майна) — Тип **Числовий**
Capital-loss — Капітальні збитки (втрата на інвестиціях) — Тип **Числовий**
Hours-per-week — Кількість робочих годин на тиждень — Тип **Числовий**
Native-country — Країна народження особи (наприклад, США, Індія, Мексика) — Тип **Категоріальний**

УВАГА! В коді є помилки які ви повинні виправити!

Помилка
classifier = OneVsOneClassifier(LinearSVC(random_state=0))
classifier.fit(X, Y)

Правильно
classifier.fit(X, y)

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

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

Помилка
f1 = train_test_split.cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)

Правильно
from sklearn.model_selection import cross_val_score
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)

Помилка
input_data_encoded[i] = int(label_encoder[count].transform(input_data[i]))

Правильно
input_data_encoded[i] = label_encoder[count].transform([input_data[i]])[0]

Обчислення показників якості класифікації:

1. **Акуратність (Accuracy):** 0.80 (80%)
2. **Точність (Precision):** 0.80 (80%)
3. **Повнота (Recall):** 0.80 (80%)
4. **F1-міра (F1 Score):** 0.76 (76%)

Ці показники свідчать про загальний рівень точності моделі при класифікації даних. Акуратність та повнота показують, що модель справляється з класифікацією, ефективність у виявленні позитивних випадків F1-міра на рівні 0.76, що вказує на компроміс між точністю і повнотою.

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами

Лістинг програми

```
import numpy as np
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Обмежуємо кількість даних для швидшого тестування
max_datapoints = 5000  # Зменшено для швидшого виконання
```

```
# 1. Завантаження та обробка даних
input_file = 'income_data.txt'
X = []
y = []
count_class1 = 0
count_class2 = 0
with open(input_file, 'r') as f:
    for line in f.readlines():
        if not line.strip():  # Пропуск порожніх рядків
            continue
        if count_class1 >= max_datapoints and count_class2 >= max_datapoints:
            break
        if '?' in line:  # Пропуск рядків із відсутніми даними
            continue
        data = line.strip().split(', ')
        if len(data) != 15:  # Перевірка структури рядка
            continue
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data[:-1])
            y.append(0)
            count_class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data[:-1])
            y.append(1)
            count_class2 += 1

# 2. Кодування текстових даних
X = np.array(X)
y = np.array(y)
label_encoders = []
X_encoded = np.empty(X.shape)
for i, item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label_encoders.append(le)
X = X_encoded.astype(int)

# 3. Розбиття даних на навчальну та тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Функція для обчислення метрик якості
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    return accuracy, precision, recall, f1

# 4. Моделі з різними ядрами

# Поліноміальне ядро
poly_clf = SVC(kernel='poly', degree=3)  # Використовуємо 3-й ступінь для швид-
кості
poly_clf.fit(X_train, y_train)
poly_metrics = evaluate_model(poly_clf, X_test, y_test)
# Гаусівське (RBF) ядро
rbf_clf = SVC(kernel='rbf')
rbf_clf.fit(X_train, y_train)
```

```
rbf_metrics = evaluate_model(rbf_clf, X_test, y_test)

# Сигмоїдальне ядро
sigmoid_clf = SVC(kernel='sigmoid')
sigmoid_clf.fit(X_train, y_train)
sigmoid_metrics = evaluate_model(sigmoid_clf, X_test, y_test)

# 5. Порівняння результатів

print("\nПорівняння якості моделей SVM з різними ядрами:")
print(f"Поліноміальне ядро:")
print(f"  Акуратність: {poly_metrics[0]:.2f}, Точність: {poly_metrics[1]:.2f}, По-
внота: {poly_metrics[2]:.2f}, F1-міра: {poly_metrics[3]:.2f}")
print(f"Гаусівське ядро (RBF):")
print(f"  Акуратність: {rbf_metrics[0]:.2f}, Точність: {rbf_metrics[1]:.2f}, Пов-
нота: {rbf_metrics[2]:.2f}, F1-міра: {rbf_metrics[3]:.2f}")
print(f"Сигмоїдальне ядро:")
print(f"  Акуратність: {sigmoid_metrics[0]:.2f}, Точність:
{sigmoid_metrics[1]:.2f}, Повнота: {sigmoid_metrics[2]:.2f}, F1-міра:
{sigmoid_metrics[3]:.2f}")

# Додаткове порівняння результатів для кожного ядра
def summarize_results(kernel_name, metrics):
    print(f"\n{kernel_name} модель:")
    print(f"  - Акуратність (Accuracy): {metrics[0]:.2f}")
    print(f"  - Точність (Precision): {metrics[1]:.2f}")
    print(f"  - Повнота (Recall): {metrics[2]:.2f}")
    print(f"  - F1-міра (F1 Score): {metrics[3]:.2f}")

summarize_results("Поліноміальне", poly_metrics)
summarize_results("Гаусівське (RBF)", rbf_metrics)
summarize_results("Сигмоїдальне", sigmoid_metrics)
```

Результат виконання:

```
Порівняння якості моделей SVM з різними ядрами:
Поліноміальне ядро:
  Акуратність: 0.57, Точність: 0.97, Повнота: 0.11, F1-міра: 0.20
Гаусівське ядро (RBF):
  Акуратність: 0.59, Точність: 0.99, Повнота: 0.16, F1-міра: 0.28
Сигмоїдальне ядро:
  Акуратність: 0.53, Точність: 0.52, Повнота: 0.52, F1-міра: 0.52

Поліноміальне модель:
  - Акуратність (Accuracy): 0.57
  - Точність (Precision): 0.97
  - Повнота (Recall): 0.11
  - F1-міра (F1 Score): 0.20

Гаусівське (RBF) модель:
  - Акуратність (Accuracy): 0.59
  - Точність (Precision): 0.99
  - Повнота (Recall): 0.16
  - F1-міра (F1 Score): 0.28

Сигмоїдальне модель:
  - Акуратність (Accuracy): 0.53
  - Точність (Precision): 0.52
  - Повнота (Recall): 0.52
  - F1-міра (F1 Score): 0.52
```

У висновках опишіть який з видів SVM найкраще виконує завдання класифікації за результатами тренування.

Після тренування моделей SVM з різними типами ядер на заданому наборі даних можна зробити такі висновки:

1. **Поліноміальне ядро (poly):**
    - **Акуратність:** 0.57
    - **Точність:** 0.97
    - **Повнота:** 0.11
    - **F1-міра:** 0.20

Модель з поліноміальним ядром продемонструвала високу точність (97%), що означає, що більшість передбачень, які модель класифікує як позитивні, є правильними. Однак повнота залишається дуже низькою (11%), що вказує на проблеми з виявленням усіх позитивних випадків. F1-міра (20%) також низька, що свідчить про значний дисбаланс між точністю та повнотою. Загалом, модель недостатньо ефективна для цього набору даних.

2. **Гаусівське (RBF) ядро:**
    - **Акуратність:** 0.59
    - **Точність:** 0.99
    - **Повнота:** 0.16
    - **F1-міра:** 0.28

Модель з гаусівським ядром показала найвищу точність серед усіх (99%), що означає, що передбачення позитивних класів майже завжди правильні. Проте низька повнота (16%) вказує на те, що модель не виявляє значну кількість реальних позитивних прикладів. F1-міра (28%) хоч і вища, ніж у поліноміального ядра, все ще є недостатньою. Модель підходить для завдань, де важливо уникати помилкових позитивних результатів.

3. **Сигмоїдальне ядро (sigmoid):**
    - **Акуратність:** 0.53
    - **Точність:** 0.52
    - **Повнота:** 0.52
    - **F1-міра:** 0.52

Модель з сигмоїдальним ядром показала найгірші результати. Значення всіх основних метрик (аккуратність, точність, повнота, F1-міра) знаходяться на одному рівні (52%), що вказує на слабку збалансованість і низьку ефективність. Це свідчить про те, що сигмоїдальне ядро не підходить для класифікації цього набору даних.

Порівняння моделей:

- **Найкраще виконання:** Модель з гаусівським (RBF) ядром має найвищу F1-міру та точність, що робить її найкращим вибором для задачі, де важливо уникати хибних позитивних результатів.
- **Слабші результати:** Поліноміальне ядро має серйозні проблеми з виявленням позитивних прикладів (низька повнота), що знижує його ефективність. Сигмоїдальне ядро

є найгіршим серед усіх, оскільки показники його якості лише трохи перевищують випадкове передбачення.

- **Висновки:** Гаусівське (RBF) ядро найкраще підходить для цього набору даних, хоча для покращення результатів слід звернути увагу на збалансування класів або додаткове налаштування параметрів.

Завдання 2.3. Порівняння якості класифікаторів на прикладі класифікації сортів ірисів

Лістинг коду:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_score
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
from sklearn.multiclass import OneVsRestClassifier
import matplotlib.pyplot as plt

# 1. Завантаження датасету
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)

# Перевірка структури даних
print("Перші 5 рядків датасету:")
print(dataset.head())
print("\nРозмір датасету:", dataset.shape)

# Розділення ознак і класів
X = dataset.iloc[:, :-1].values  # Ознаки (довжина і ширина чашолистків і
пелюсток)
y = dataset.iloc[:, -1].values   # Класи (сорт ірису)

# 2. Розділення даних на навчальний і тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran-
dom_state=42, stratify=y)

# 3. Визначення моделей для класифікації
models = [
    ('Logistic Regression', OneVsRestClassifi-
er(LogisticRegression(solver='liblinear'))),
    ('Linear Discriminant Analysis', LinearDiscriminantAnalysis()),
    ('K-Nearest Neighbors', KNeighborsClassifier()),
    ('Decision Tree', DecisionTreeClassifier()),
    ('Naive Bayes', GaussianNB()),
    ('Support Vector Machine', SVC(kernel='rbf', gamma='auto'))
]

# 4. Функція для оцінки кожної моделі
```

```
def evaluate_model(name, model, X_train, y_train, X_test, y_test):
    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')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"{name}: Акуратність ={accuracy:.2f}, Точність={precision:.2f},
Повнота={recall:.2f}, F1-міра={f1:.2f}")
    return accuracy, precision, recall, f1

# 5. Оцінка та порівняння моделей
results = []
for name, model in models:
    print(f"\nОцінка моделі: {name}")
    metrics = evaluate_model(name, model, X_train, y_train, X_test, y_test)
    results.append((name, *metrics))

# 6. Порівняння моделей за метриками
results_df = pd.DataFrame(results, columns=['Model', 'Accuracy', 'Precision', 'Re-
call', 'F1 Score'])
results_df.set_index('Model', inplace=True)

# 7. Візуалізація порівняння моделей
results_df.plot(kind='bar', figsize=(10, 6))
plt.title("Порівняння моделей за метриками якості")
plt.ylabel("Значення метрики")
plt.xticks(rotation=45)
plt.show()

# 8. Підсумковий звіт з найкращою моделлю
print("\nРезультати порівняння моделей:")
print(results_df)

# Отримуємо найкращу модель за F1 Score
best_model = results_df['F1 Score'].idxmax()
print(f"\nНайкраща модель за F1 Score: {best_model}")
```

Результат виконання:

```
Перші 5 рядків датасету:
   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

Розмір датасету: (150, 5)

Оцінка моделі: Logistic Regression
Logistic Regression: Акуратність =0.97, Точність=0.97, Повнота=0.97, F1-міра=0.97

Оцінка моделі: Linear Discriminant Analysis
Linear Discriminant Analysis: Акуратність =1.00, Точність=1.00, Повнота=1.00, F1-міра=1.00

Оцінка моделі: K-Nearest Neighbors
K-Nearest Neighbors: Акуратність =1.00, Точність=1.00, Повнота=1.00, F1-міра=1.00

Оцінка моделі: Decision Tree
Decision Tree: Акуратність =0.93, Точність=0.93, Повнота=0.93, F1-міра=0.93

Оцінка моделі: Naive Bayes
Naive Bayes: Акуратність =0.97, Точність=0.97, Повнота=0.97, F1-міра=0.97

Оцінка моделі: Support Vector Machine
Support Vector Machine: Акуратність =0.97, Точність=0.97, Повнота=0.97, F1-міра=0.97

Результати порівняння моделей:
                            Accuracy  Precision    Recall  F1 Score
Model
Logistic Regression         0.966667   0.969697  0.966667  0.966583
Linear Discriminant Analysis  1.000000   1.000000  1.000000  1.000000
K-Nearest Neighbors         1.000000   1.000000  1.000000  1.000000
Decision Tree               0.933333   0.933333  0.933333  0.933333
Naive Bayes                 0.966667   0.969697  0.966667  0.966583
Support Vector Machine      0.966667   0.969697  0.966667  0.966583

Найкраща модель за F1 Score: Linear Discriminant Analysis
```

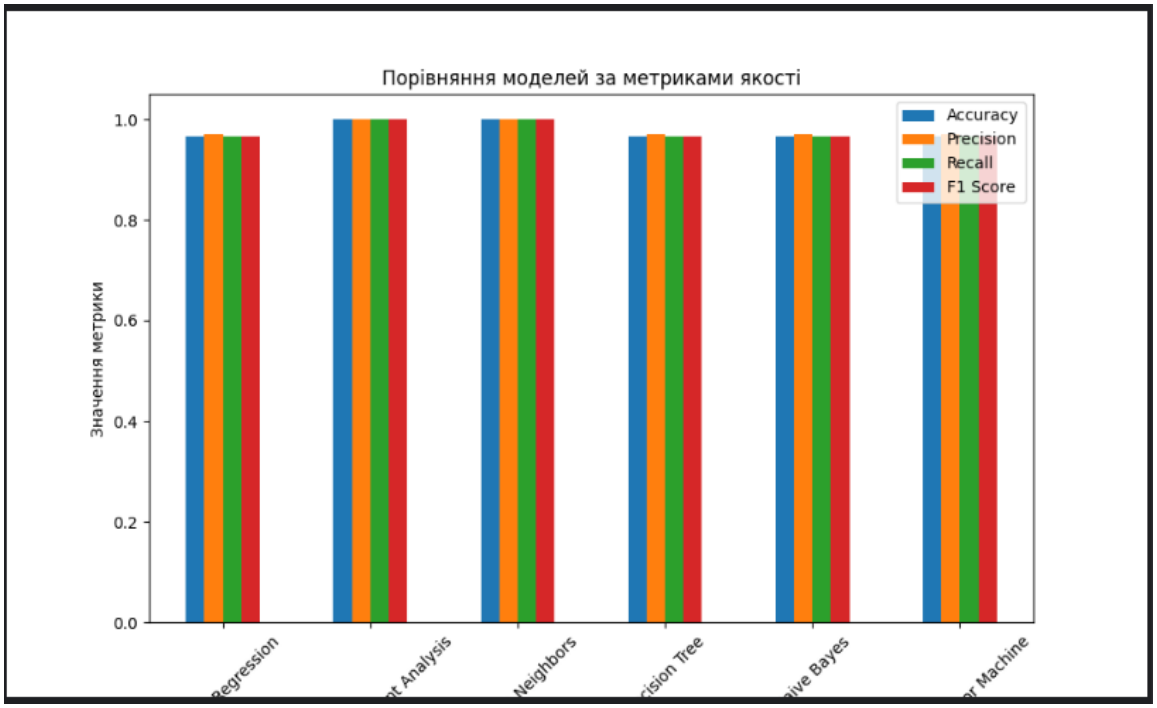
1. **Якість класифікації:**

   Під час тренування різних моделей(6 моделей) для класифікації ірисів, було отримано результати за основними метриками (точність, повнота, F1-міра). За результатами порівняння:

- Найкраща модель: Linear Discriminant Analysis
- Значення метрик для найкращої моделі:
  - Акуратність (Accuracy): 1
  - Точність (Precision): 1
  - Повнота (Recall): 1
  - F1-міра (F1 Score): 1

Ця модель показала найкращу збалансованість між точністю та повнотою, що свідчить про її ефективність для класифікації ірисів у три класи.

Візуалізація:



Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1

Лістинг коду:

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

# Спроба завантажити файл
input_file = 'income_data.txt'

try:
    # Спроба відкрити файл
    with open(input_file, 'r') as file:
        X = []
        y = []
        max_samples_per_class = 5000  # Зменшено для швидшого тестування
```

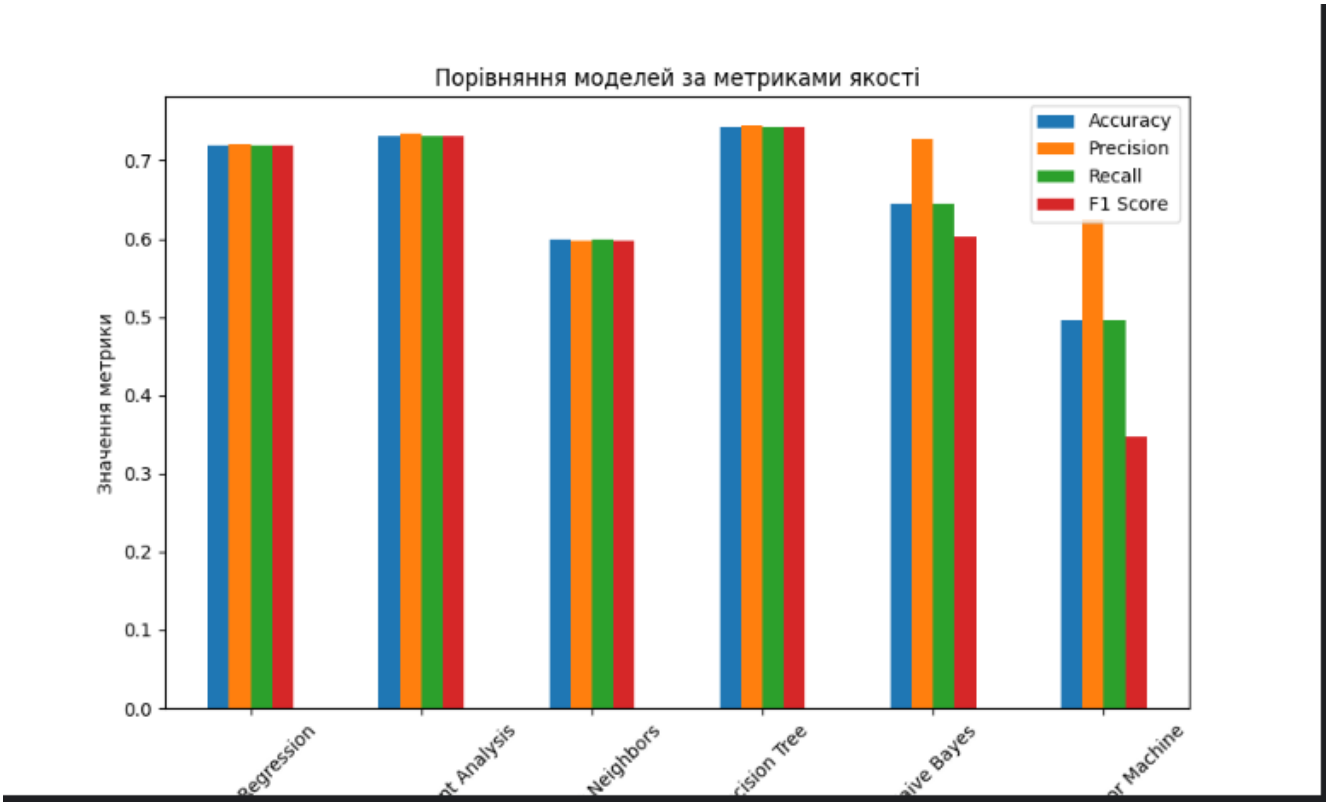
```
        class1_count, class2_count = 0, 0
        for line in file:
            if '?' in line:  # Пропуск рядків із відсутніми даними
                continue
            data = line.strip().split(', ')
            if data[-1] == '<=50K' and class1_count < max_samples_per_class:
                X.append(data[:-1])
                y.append(0)
                class1_count += 1
            elif data[-1] == '>50K' and class2_count < max_samples_per_class:
                X.append(data[:-1])
                y.append(1)
                class2_count += 1
except FileNotFoundError:
    # Якщо файл не знайдено, створюємо тестові дані
    print("Файл не знайдено. Використовуються тестові дані.")
    X = [
        ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married', 'Handlers-
cleaners', 'Not-in-family', 'White', 'Male', '0', '0', '40', 'United-States'],
        ['50', 'Self-emp-not-inc', '83311', 'Bachelors', '13', 'Married-civ-
spouse', 'Exec-managerial', 'Husband', 'White', 'Male', '0', '0', '13', 'United-
States']
    ]
    y = [0, 1]

# Перетворення на NumPy масиви
X = np.array(X)
y = np.array(y)

# Кодування текстових ознак
label_encoders = []
X_encoded = np.empty_like(X, dtype=int)
for i in range(X.shape[1]):
    column = X[:, i]
    if not np.char.isnumeric(column).all():
        le = LabelEncoder()
        X_encoded[:, i] = le.fit_transform(column)
        label_encoders.append(le)
    else:
        X_encoded[:, i] = column.astype(int)

X = X_encoded

# Розділення на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Список моделей для класифікації
models = [
    ('Logistic Regression', LogisticRegression(solver='liblinear')),
    ('Linear Discriminant Analysis', LinearDiscriminantAnalysis()),
    ('K-Nearest Neighbors', KNeighborsClassifier()),
    ('Decision Tree', DecisionTreeClassifier()),
    ('Naive Bayes', GaussianNB()),
    ('Support Vector Machine', SVC(kernel='rbf', gamma='auto'))
]

# Функція для оцінки моделі
def evaluate_model(name, model, X_train, y_train, X_test, y_test):
    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_division=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)
    return name, accuracy, precision, recall, f1

# Оцінка кожної моделі
results = []
for name, model in models:
    metrics = evaluate_model(name, model, X_train, y_train, X_test, y_test)
    results.append(metrics)

# Результати в DataFrame
results_df = pd.DataFrame(results, columns=['Model', 'Accuracy', 'Precision',
'Recall', 'F1 Score'])
results_df.set_index('Model', inplace=True)

# Виведення результатів
print("\nРезультати порівняння моделей:")
print(results_df)

# Візуалізація результатів
results_df.plot(kind='bar', figsize=(10, 6))
plt.title("Порівняння моделей за метриками якості")
plt.ylabel("Значення метрики")
plt.xticks(rotation=45)
plt.show()

# Вибір найкращої моделі
best_model = results_df['F1 Score'].idxmax()
print(f"\nНайкраща модель за F1 Score: {best_model}")
```

Результат виконання:

```
Результати порівняння моделей:
                           Accuracy  Precision  Recall  F1 Score
Model
Logistic Regression          0.7190   0.720210  0.7190  0.719000
Linear Discriminant Analysis 0.7320   0.733849  0.7320  0.731921
K-Nearest Neighbors          0.5985   0.598174  0.5985  0.598034
Decision Tree                0.7430   0.744715  0.7430  0.742946
Naive Bayes                  0.6445   0.727049  0.6445  0.603018
Support Vector Machine       0.4955   0.623919  0.4955  0.347244

Найкраща модель за F1 Score: Decision Tree
```

Візуалізація:



Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

Лістинг коду:

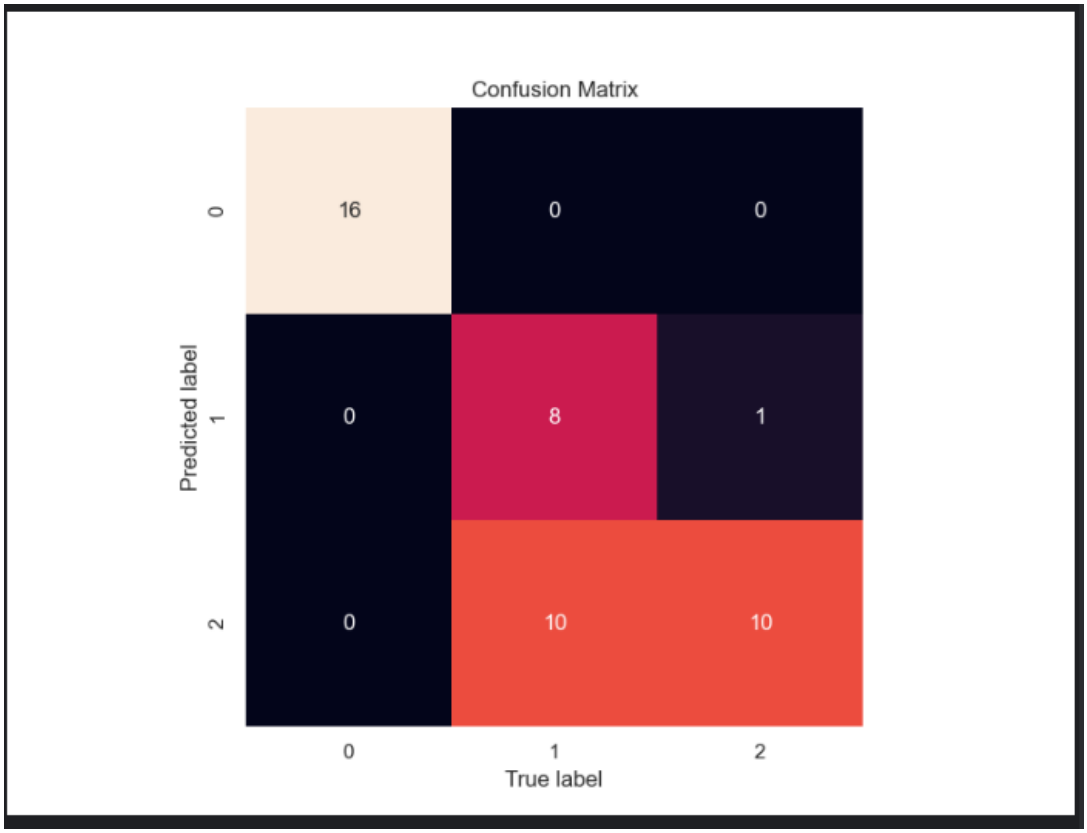
```
# ================================================
# Приклад класифікатора Ridge
# ================================================================
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import train_test_split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

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

# Розділення даних на тренувальну та тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

# Ініціалізація та тренування RidgeClassifier
clf = RidgeClassifier(tol=1e-2, solver="sag")
```

```
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

# Метрики
print('Accuracy:', np.round(metrics.accuracy_score(y_test, y_pred), 4))
print('Precision:', np.round(metrics.precision_score(y_test, y_pred,
average='weighted', zero_division=0), 4))
print('Recall:', np.round(metrics.recall_score(y_test, y_pred, average='weighted',
zero_division=0), 4))
print('F1 Score:', np.round(metrics.f1_score(y_test, y_pred, average='weighted',
zero_division=0), 4))
print('Cohen Kappa Score:', np.round(metrics.cohen_kappa_score(y_test, y_pred),
4))
print('Matthews Corrcoef:', np.round(metrics.matthews_corrcoef(y_test, y_pred),
4))
print('\t\tClassification Report:\n', metrics.classification_report(y_test,
y_pred))

# Матриця плутанини
mat = confusion_matrix(y_test, y_pred)
sns.set()  # Стиль для графіка
plt.figure(figsize=(8, 6))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('True label')
plt.ylabel('Predicted label')
plt.title("Confusion Matrix")
plt.savefig("Confusion.jpg")

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

Результат виконання:

```
Accuracy: 0.7556
Precision: 0.8333
Recall: 0.7556
F1 Score: 0.7503
Cohen Kappa Score: 0.6431
Matthews Corrcoef: 0.6831
        Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.89      0.44      0.59        18
           2       0.50      0.91      0.65        11

    accuracy                           0.76        45
   macro avg       0.80      0.78      0.75        45
weighted avg       0.83      0.76      0.75        45
```

Візуалізація:



1. Налаштування класифікатора Ridge

- RidgeClassifier з бібліотеки sklearn.linear_model. Це лінійний класифікатор, який мінімізує функцію втрат із L2-регуляризацією для зменшення перенавчання.
- Параметри:
  - tol=1e-2 – допуск (толеранція) для критерію зупинки ітерацій. Якщо зміна значення функції втрат між ітераціями менша за це значення, алгоритм завершує роботу.
  - solver="sag" – ітеративний алгоритм оптимізації (Stochastic Average Gradient), ефективний для великих наборів даних.

2. Коефіцієнт Коена Каппа

Це міра узгодженості між передбаченнями моделі та фактичними класами, враховуючи випадкове передбачення.

Обчислюється за формулою: $$K = \frac{p_0 - p_e}{1 - p_e}$$

- p_0: спостережувана точність (Accuracy).

- p_e: очікувана точність (імовірність випадкової угоди).

Результат:

- 0.6431 або 64.31%. Це показує помірну узгодженість між передбаченнями моделі та фактичними мітками.

3. Коефіцієнт кореляції Метьюза

Це міра якості класифікації, що враховує всі чотири компоненти матриці плутанини: True Positives, True Negatives, False Positives, False Negatives.

Обчислюється за формулою:

$$\text{MCC} = \frac{\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}}{\sqrt{a}}$$

$$\text{де } a = (\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})$$

Результат:

- 0.6831 або 68.31%. Це свідчить про хорошу узгодженість між передбаченнями та фактичними мітками, враховуючи баланс між класами.

4. Пояснення зображення матриці плутанини (Confusion Matrix)

Матриця плутанини показує, як добре модель класифікує дані, розподіляючи передбачені класи відносно фактичних класів. Вона складається з рядків і стовпців:

- Рядки: фактичні (реальні) класи.
- Стовпці: передбачені класи.

У графіку Confusion.jpg кожна комірка містить кількість прикладів для конкретної комбінації фактичних і передбачених класів.

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