ЛАБОРАТОРНА РОБОТА № 2

ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ

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

Варіант 1

Хід роботи:

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

Назва ознаки	**Тип даних**	**Опис**
age	Цілочисельний	Вік особи в роках
workclass	Категоріальний	Тип зайнятості (наприклад, приватний сектор, державна служба, самозайнятість тощо)
fnlwgt	Цілочисельний	Вага випадку, що вказує на кількість людей у популяції, яких представляє цей запис
education	Категоріальний	Рівень освіти (наприклад, бакалавр, середня школа, магістр тощо)
education-num	Цілочисельний	Кількісне представлення рівня освіти
marital-status	Категоріальний	Сімейний стан (наприклад, одружений, розлучений, ніколи не був одружений тощо)
occupation	Категоріальний	Професія (наприклад, технічна підтримка, продажі, управління тощо)
relationship	Категоріальний	Сімейні відносини (наприклад, чоловік, дружина, дитина, не в сім'ї тощо)
race	Категоріальний	Раса (наприклад, білий, азіатсько-тихоокеанський острів'янин, афроамериканець тощо)
sex	Категоріальний	Стать (чоловік або жінка)
capital-gain	Цілочисельний	Прибуток від капіталу, отриманий за рік
capital-loss	Цілочисельний	Втрата капіталу за рік
hours-per-week	Цілочисельний	Середня кількість робочих годин на тиждень
native-country	Категоріальний	Країна народження (наприклад, Сполучені Штати, Канада, Німеччина тощо)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import LinearSVC, SVC
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

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

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

# Відкриття файлу і читання рядків
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[:-1].split(', ')

        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1

        if 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]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])
```

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

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


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

# Навчання класифікатора
classifier.fit(X_train, y_train)

# Прогнозування результатів для тестового набору
y_test_pred = classifier.predict(X_test)

# Обчислення F1-міри для SVM-класифікатора
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("F1 score: " + str(round(100 * f1.mean(), 2)) + "%")

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

# Кодування тестової точки даних
input_data_encoded = [-1] * len(input_data)
count = 0
for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = int(input_data[i])
    else:
        input_data_encoded[i] =
int(label_encoder[count].transform([input_data[i]])[0])
        count += 1

input_data_encoded = np.array(input_data_encoded)

# Використання класифікатора для кодованої точки даних
# та виведення результату
predicted_class = classifier.predict([input_data_encoded])
print(label_encoder[-1].inverse_transform(predicted_class)[0])

accuracy = accuracy_score(y_test, y_test_pred)
print("Accuracy:" + str(round(100 * accuracy, 2)) + "%")

precision = precision_score(y_test, y_test_pred, average="weighted")
print("Precision:" + str(round(100 * precision, 2)) + "%")
```

```
recall = recall_score(y_test, y_test_pred, average="weighted")
print("Recall:" + str(round(100 * recall, 2)) + "%")
```

```
F1 score: 76.09%
<=50K
Accuracy:79.56%
Precision:79.26%
Recall:79.56%
```

Рис. 1. Результати класифікатора щорічного прибутку

Висновок: Точка належить до категорії доходу нижче 50 тисяч із точністю 79,56%.

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами
Модифікований код для аналізу нелінійних класифікаторів SVM: перший використовує поліноміальне ядро, другий — гауссове, а третій — сигмоїдальне.

```
classifier = OneVsOneClassifier(SVC(kernel='rbf', random_state=0))
classifier = OneVsOneClassifier(SVC(kernel='sigmoid', random_state=0))
classifier = OneVsOneClassifier(SVC(kernel='poly', random_state=0))
```

```
F1 score: 71.95%
<=50K
Accuracy:78.19%
Precision:82.82%
Recall:78.19%
```

```
F1 score: 63.77%
<=50K
Accuracy:60.47%
Precision:60.64%
Recall:60.47%
```

```
F1 score: 63.77%
<=50K
Accuracy:60.47%
Precision:60.64%
Recall:60.47%
```

Висновок: Результати демонструють, що гауссове ядро є найефективнішим для даної задачі, оскільки воно забезпечує найвищі значення точності, F1 Score, Precision та Recall. Це підтверджує, що класифікатор із гауссовим ядром краще адаптується до розподілу даних і враховує складні нелінійні взаємозв'язки, що робить його найкращим вибором.

Завдання 2.3. Порівняння якості класифікаторів на прикладі класифікації сортів ірисів.

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
print("Ключі iris_dataset: \n{}".format(iris_dataset.keys()))
print(iris_dataset['DESCR'][:193] + "\n...")
print("Назви відповідей: {}".format(iris_dataset['target_names']))
print("Назва ознак: \n{}".format(iris_dataset['feature_names']))
print("Тип масиву data: {}".format(type(iris_dataset['data'])))
print("Форма масиву data: {}".format(iris_dataset['data'].shape))
print("Значення ознак для перших 5 прикладів", iris_dataset['data'][:5])
print("Тип масиву target:{}".format(type(iris_dataset['target'])))
print("Відповіді:\n{}".format(iris_dataset['target']))
```

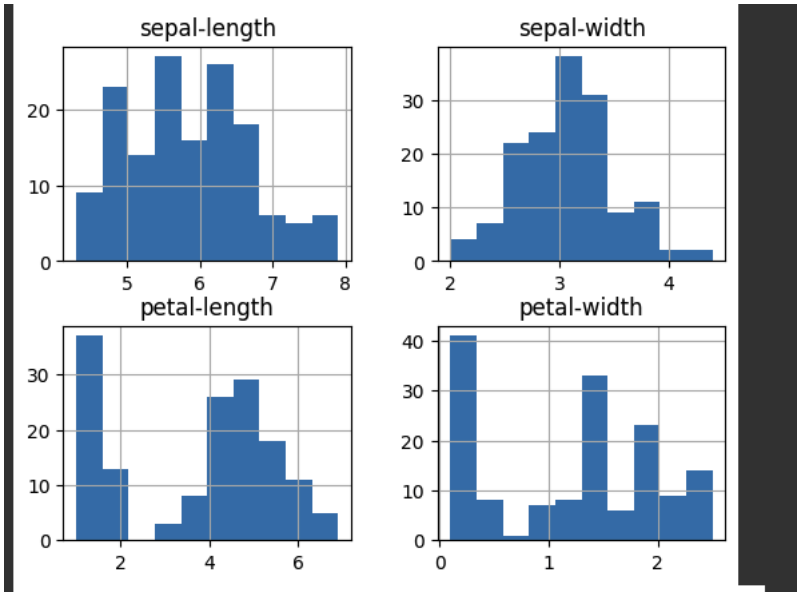
```
Ключі iris_dataset:
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive
...
Назви відповідей: ['setosa' 'versicolor' 'virginica']
Назва ознак:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Тип масиву data: <class 'numpy.ndarray'>
Форма масиву data: (150, 4)
Значення ознак для перших 5 прикладів [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
Тип масиву target:<class 'numpy.ndarray'>
Відповіді:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 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 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]
```

```
import numpy as np
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

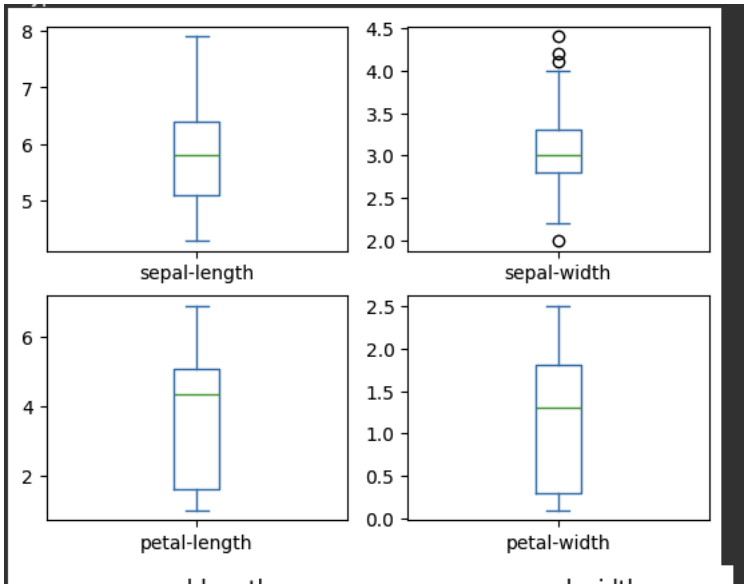
```
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',
'class']
dataset = read_csv(url, names=names)

# shape
print(dataset.shape)

# Зріз даних head
print(dataset.head(20))

# Стастичні зведення методом describe
print(dataset.describe())

# Розподіл за атрибутом class
print(dataset.groupby('class').size())

# Діаграма розмаху
dataset.plot(kind='box', subplots=True, layout=(2,2),
sharex=False, sharey=False)
pyplot.show()

# Гістограма розподілу атрибутів датасета
dataset.hist()
pyplot.show()

#Матриця діаграм розсіювання
scatter_matrix(dataset)
pyplot.show()

# Розділення датасету на навчальну та контрольну вибірки
array = dataset.values
# Вибір перших 4-х стовпців
X = array[:,0:4]
# Вибір 5-го стовпця
Y = array[:,4]
# Разделение X и y на обучающую и контрольную выборки
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y,
test_size=0.20, random_state=1)

# Завантажуємо алгоритми моделі
models = []
models.append(('LR', LogisticRegression(solver='liblinear',
multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
# оцінюємо модель на кожній ітерації
results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
```

```
scoring='accuracy')
    results.append(cv_results)
    names.append(name)

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

# Порівняння алгоритмів
pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()

# Створюємо прогноз на контрольній вибірці
model = SVC(gamma='auto')
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

# Оцінюємо прогноз
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))

# Завантажуємо набір даних Iris
iris_dataset = load_iris()

# Розділяємо дані на навчальну та тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
iris_dataset['target'], random_state=0)

# Створюємо і тренуємо модель KNN
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

# Нові дані для прогнозу
X_new = np.array([[5, 2.9, 1, 0.2]])
print("Форма масиву X_new: {}".format(X_new.shape))

# Робимо прогноз
prediction = knn.predict(X_new)
print("Прогноз: {}".format(prediction))
print("Спрогнозована мітка: {}".format(iris_dataset['target_names'][prediction]))
```

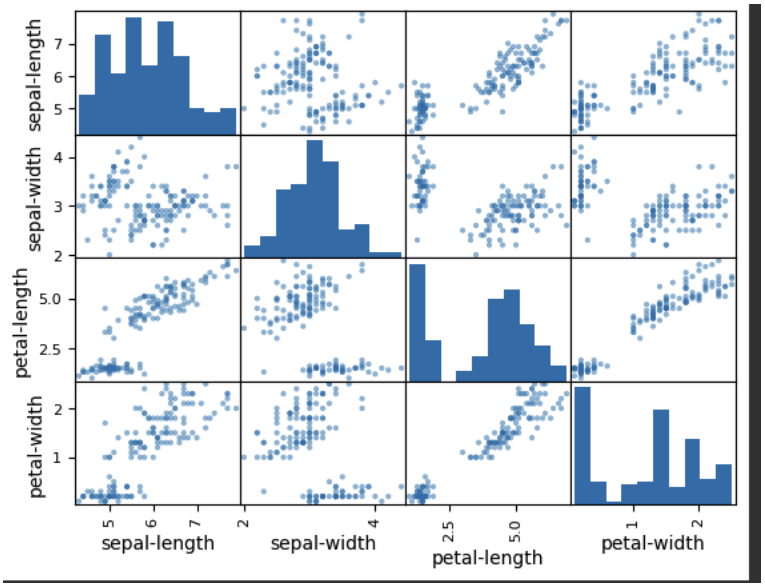
```
(150, 5)
     sepal-length  sepal-width  petal-length  petal-width        class
0             5.1          3.5           1.4          0.2  Iris-setosa
1             4.9          3.0           1.4          0.2  Iris-setosa
2             4.7          3.2           1.3          0.2  Iris-setosa
3             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
       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
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```



Гістограма розподілу атрибутів датасета.

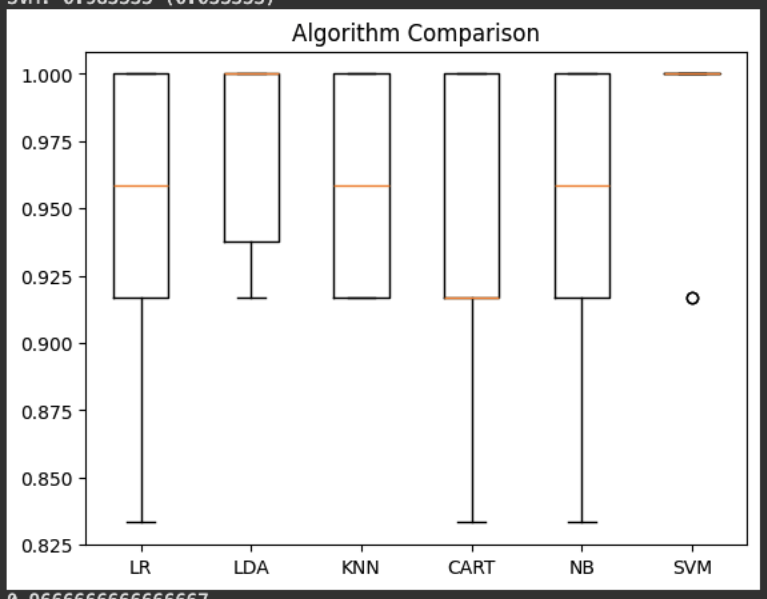


Діаграма розмаху



Матриця діаграм розсіювання

```
LR: 0.941667 (0.065085)
LDA: 0.975000 (0.038188)
KNN: 0.958333 (0.041667)
CART: 0.941667 (0.053359)
NB: 0.950000 (0.055277)
SVM: 0.983333 (0.033333)
```




```
0.9666666666666667
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
                 precision    recall  f1-score   support

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

       accuracy                           0.97        30
      macro avg       0.95      0.97      0.96        30
   weighted avg       0.97      0.97      0.97        30
```

Метод опорних векторів (SVM) показав найвищу середню точність у 98,33% з мінімальним стандартним відхиленням, перевершуючи лінійний дискримінантний аналіз (LDA). Хоча LDA поступається за точністю, він успішно класифікував квітку з параметрами [5, 2.9, 1, 0.2] як "setosa".

Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1

```
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, 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

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

# Читання даних
```

```
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000

# Відкриття файлу і читання рядків
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[:-1].split(', ')

        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1

        if 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]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])

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

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

# Навчання та оцінка логістичної регресії
lr_model = LogisticRegression(max_iter=10000)
lr_model.fit(X_train, y_train)
lr_predictions = lr_model.predict(X_test)
print('Logistic Regression Accuracy:', accuracy_score(y_test, lr_predictions))
print('Logistic Regression Precision:', precision_score(y_test, lr_predictions,
average='weighted'))
print('Logistic Regression Recall:', recall_score(y_test, lr_predictions,
average='weighted'))
print('Logistic Regression F1 Score:', f1_score(y_test, lr_predictions,
average='weighted'))
print("\n")

# Навчання та оцінка моделі лінійного дискримінантного аналізу
lda_model = LinearDiscriminantAnalysis()
lda_model.fit(X_train, y_train)
lda_predictions = lda_model.predict(X_test)
print('Linear Discriminant Accuracy:', accuracy_score(y_test, lda_predictions))
```

```
print('Linear Discriminant Precision:', precision_score(y_test, lda_predictions,
average='weighted'))
print('Linear Discriminant Recall:', recall_score(y_test, lda_predictions,
average='weighted'))
print('Linear Discriminant F1 Score:', f1_score(y_test, lda_predictions,
average='weighted'))
print("\n")

# Навчання та оцінка моделі k-ближчих сусідів
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)
print('K-Nearest Neighbors Accuracy:', accuracy_score(y_test, knn_predictions))
print('K-Nearest Neighbors Precision:', precision_score(y_test, knn_predictions,
average='weighted'))
print('K-Nearest Neighbors Recall:', recall_score(y_test, knn_predictions,
average='weighted'))
print('K-Nearest Neighbors F1 Score:', f1_score(y_test, knn_predictions,
average='weighted'))
print("\n")

# Навчання та оцінка дерева рішень
cart_model = DecisionTreeClassifier()
cart_model.fit(X_train, y_train)
cart_predictions = cart_model.predict(X_test)
print('Decision Tree Accuracy:', accuracy_score(y_test, cart_predictions))
print('Decision Tree Precision:', precision_score(y_test, cart_predictions,
average='weighted'))
print('Decision Tree Recall:', recall_score(y_test, cart_predictions,
average='weighted'))
print('Decision Tree F1 Score:', f1_score(y_test, cart_predictions,
average='weighted'))
print("\n")

# Навчання та оцінка моделі наївного Баєса
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_predictions = nb_model.predict(X_test)
print('Naive Bayes Accuracy:', accuracy_score(y_test, nb_predictions))
print('Naive Bayes Precision:', precision_score(y_test, nb_predictions,
average='weighted'))
print('Naive Bayes Recall:', recall_score(y_test, nb_predictions,
average='weighted'))
print('Naive Bayes F1 Score:', f1_score(y_test, nb_predictions,
average='weighted'))
print("\n")

# Навчання та оцінка методу опорних векторів
svm_model = SVC()
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
print('Support Vector Machine Accuracy:', accuracy_score(y_test, svm_predictions))
print('Support Vector Machine Precision:', precision_score(y_test,
svm_predictions, average='weighted'))
print('Support Vector Machine Recall:', recall_score(y_test, svm_predictions,
average='weighted'))
print('Support Vector Machine F1 Score:', f1_score(y_test, svm_predictions,
average='weighted'))
print("\n")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic Regression Accuracy: 0.8000994530084535
Logistic Regression Precision: 0.7863757844954583
Logistic Regression Recall: 0.8000994530084535
Logistic Regression F1 Score: 0.778308035199406


Linear Discriminant Accuracy: 0.8112050389524283
Linear Discriminant Precision: 0.7995573387713292
Linear Discriminant Recall: 0.8112050389524283
Linear Discriminant F1 Score: 0.7949351919633366


K-Nearest Neighbors Accuracy: 0.7677772252610642
K-Nearest Neighbors Precision: 0.7431293244415015
K-Nearest Neighbors Recall: 0.7677772252610642
K-Nearest Neighbors F1 Score: 0.7427191712661916


Decision Tree Accuracy: 0.8054036134593071
Decision Tree Precision: 0.8086158854965152
Decision Tree Recall: 0.8054036134593071
Decision Tree F1 Score: 0.8068769150394566


Naive Bayes Accuracy: 0.7894911321067463
Naive Bayes Precision: 0.7743364987007835
Naive Bayes Recall: 0.7894911321067463
Naive Bayes F1 Score: 0.7590556977541115


Support Vector Machine Accuracy: 0.7818664014586442
Support Vector Machine Precision: 0.8281852616923893
Support Vector Machine Recall: 0.7818664014586442
Support Vector Machine F1 Score: 0.7151154420230494
```

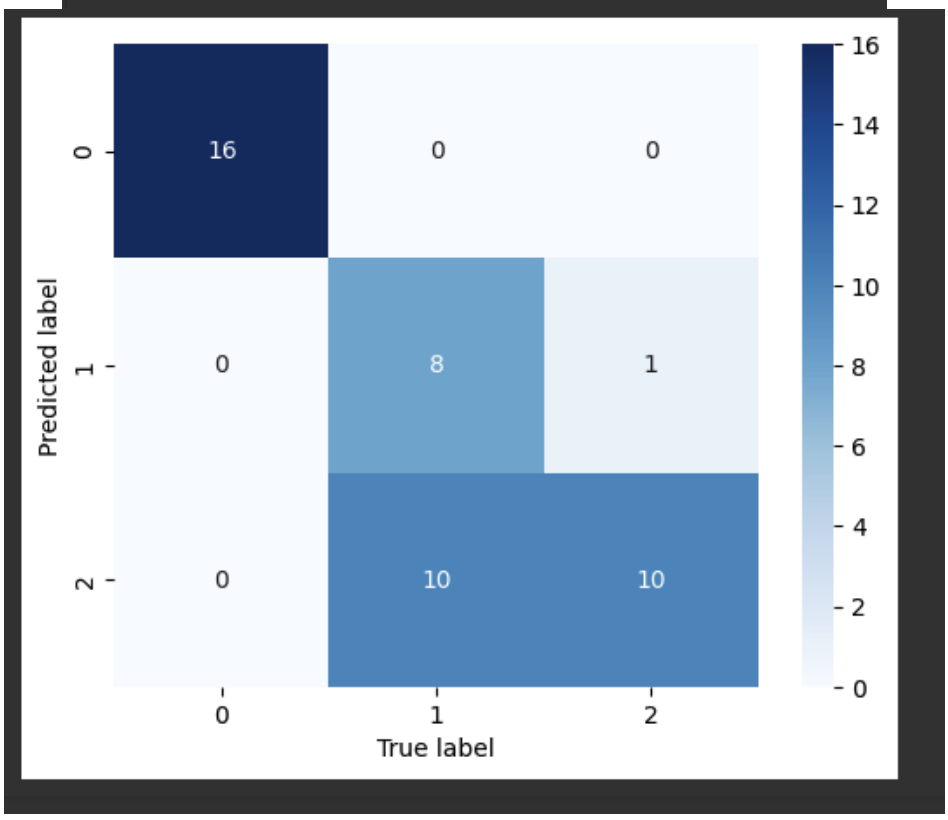
Аналіз метрик показав, що модель Linear Discriminant Analysis (LDA) є найкращим вибором для цієї задачі, оскільки вона досягає найвищих значень Accuracy, Recall та F1 Score, забезпечуючи стабільні й збалансовані результати класифікації.

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import RidgeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
recall_score, f1_score, cohen_kappa_score, matthews_corrcoef,
classification_report
from io import BytesIO
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

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

y_pred = clf.predict(X_test)

print('Accuracy:', np.round(accuracy_score(y_test, y_pred), 4))
print('Precision:', np.round(precision_score(y_test, y_pred, average='weighted'),
4))
print('Recall:', np.round(recall_score(y_test, y_pred, average='weighted'), 4))
print('F1 Score:', np.round(f1_score(y_test, y_pred, average='weighted'), 4))
```

```
print('Cohen Kappa Score:', np.round(cohen_kappa_score(y_test, y_pred), 4))
print('Matthews Corrcoef:', np.round(matthews_corrcoef(y_test, y_pred), 4))
print('\t\tClassification Report:\n', classification_report(y_test, y_pred))

mat = confusion_matrix(y_test, y_pred)

sns.heatmap(mat.T, square=True, annot=True, fmt='d', cmap="Blues")
plt.xlabel('True label')
plt.ylabel('Predicted label')
plt.savefig("Confusion.jpg")

f = BytesIO()
plt.savefig(f, format="svg")
plt.show()
```

```
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

Для класифікатора Ridge було застосовано наступні параметри:

- **tol=1e-2**: цей параметр визначає допустиму зміну значення, при якій алгоритм припиняє ітерації. У даному випадку значення 1e-2 (або 0.01) означає, що алгоритм завершує навчання, коли зміни стають меншими за 0.01, що сприяє швидшій конвергенції.

- **solver="sag"**: параметр вказує метод оптимізації, який використовується для класифікації. У цьому випадку обрано SAG (Stochastic Average Gradient), оптимальний для роботи з великими наборами даних, завдяки ефективній обробці.

Результати та метрики якості моделі:

1. **Точність (Accuracy)**: 0.7556 — частка правильно класифікованих прикладів серед усіх зразків.
2. **Прецизійність (Precision)**: 0.8333 — середня точність передбачень для позитивних класів, яка показує, скільки з передбачених позитивних випадків є правильними.
3. **Чутливість (Recall)**: 0.7556 — доля коректно ідентифікованих позитивних зразків серед усіх фактичних позитивів.
4. **F1-міра**: 0.7503 — гармонійне середнє між точністю та чутливістю, що відображає баланс між цими метриками.
5. **Коефіцієнт Каппа (Cohen's Kappa)**: 0.6431 — метрика узгодженості між передбаченнями моделі та реальними значеннями, враховуючи випадкові збіги.
6. **Коефіцієнт Метьюса (Matthews Correlation Coefficient)**: 0.6831 — показник, що оцінює якість класифікації, враховуючи баланс між класами. Значення варіюються від -1 (повна невідповідність) до 1 (ідеальна відповідність).

- **Коефіцієнт Каппа Коена**: відображає узгодженість між передбаченнями моделі та фактичними значеннями. Значення 1 свідчить про повну відповідність, 0 — про випадкові збіги, а негативні значення вказують на розбіжності. У даному випадку метрика демонструє, наскільки ефективно модель розпізнає класи з урахуванням випадковості.
- **Коефіцієнт кореляції Метьюса**: оцінює якість класифікації на основі всіх елементів матриці заплутаності. Цей показник ефективно враховує баланс між позитивними та негативними класами, де значення 1 означає ідеальне передбачення, 0 — випадковий результат, а -1 — повну розбіжність.

Посилання на Github:
https://github.com/Vladislaw2533/SHI_Barabash_Vlad_IPZ_21_3

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