

## 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.

### 2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ

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

**14 ознак з файлу `adult.names` - їх назви та що вони позначають та вид.**

1. **age** – числова (continuous).
2. **workclass** – категоріальна: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
3. **fnlwgt** – числова (continuous).
4. **education** – категоріальна: Bachelors, Some-college, HS-grad, Prof-school, та інші.
5. **education-num** – числова (continuous), кількість років освіти.
6. **marital-status** – категоріальна: Married-civ-spouse, Divorced, Never-married, та інші.
7. **occupation** – категоріальна: Tech-support, Sales, Exec-managerial, Prof-specialty, та інші.
8. **relationship** – категоріальна: Wife, Own-child, Husband, Not-in-family, та інші.
9. **race** – категоріальна: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
10. **sex** – категоріальна: Female, Male.
11. **capital-gain** – числова (continuous).

12. **capital-loss** – числова (continuous).

13. **hours-per-week** – числова (continuous).

14. **native-country** – категоріальна: United-States, Cambodia, England, та інші.

### 2.1.1. Ознайомтесь з набором даних.

```
✓ [90] import numpy as np
0a      import matplotlib.pyplot as plt
      from sklearn import preprocessing
      from sklearn.svm import LinearSVC
      from sklearn.multiclass import OneVsOneClassifier
      from sklearn.model_selection import train_test_split
      from sklearn.model_selection import cross_val_score

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

```
✓ [91] # Читання даних
0a      X = []
      y = []
      count_class1 = 0
      count_class2 = 0
      max_datapoints = 25000
```

```
✓ [92] with open(input_file, 'r') as f:
0a      for line in f:
          if count_class1 >= max_datapoints and count_class2 >= max_datapoints:
              break
          if '?' in line:
              continue

          data = line.strip().split(',')

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

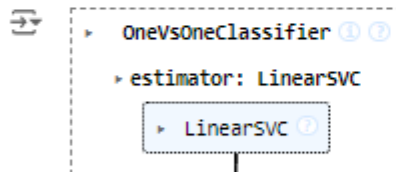
```
✓ [93] # Перетворення на масив numpy
0s X = np.array(X)

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

for i,item in enumerate(X[1]):
    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)
```

```
✓ [94] # Створення SVM-класифікатора
0s classifier = OneVsOneClassifier(LinearSVC(random_state=0))
```

```
✓ [95] # Навчання класифікатора
0s classifier.fit(X, y)
```



```
✓ [96] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)
```

```
✓ [97] classifier = OneVsOneClassifier(LinearSVC(random_state=0))
0s classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)
```

```
✓ [98] # Обчислення F-міри для SVM-класифікатора
0s f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("F1 score: " + str(round(100*f1.mean(), 2)) + "%")
```

F1 score: 74.26%

```

[100] # Передбачення результату для тестової точки даних
input_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Nevermarried', 'Handlers-cleaners', 'Not-in-family', 'White', 'Male',
'0', '0', '40', 'United-States']

[107] input_data_encoded = [-1] * len(input_data)
count = 0

for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = int(item)
    else:
        try:
            input_data_encoded[i] = int(label_encoder[count].transform([item])[0])
        except ValueError:
            # Якщо значення не зустрічалося раніше, присвоюємо значення -1 або інше значення для "private"\ "unknown"
            input_data_encoded[i] = -1
        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.reshape(1, -1))
print(label_encoder[-1].inverse_transform(predicted_class)[0])

```

≤50K

**Обчисліть значення інших показників якості класифікації (акуратність, повнота, точність) та разом з F1 занесіть їх у звіт. (Див. ЛР-1).**

```

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

# Передбачаємо результати для тестової вибірки
y_pred = classifier.predict(X_test)

# Обчислюємо метрики
accuracy = round(accuracy_score(y_test, y_pred), 2)
precision = round(precision_score(y_test, y_pred, average='weighted'), 2)
recall = round(recall_score(y_test, y_pred, average='weighted'), 2)
f1 = round(f1_score(y_test, y_pred, average='weighted'), 2)

# Виведення результатів
print(f"Точність (accuracy): {accuracy}%")
print(f"прецизійність (precision): {precision}%")
print(f"повнота (recall): {recall}%")
print(f"F1-score: {f1}%")

```

Точність (accuracy): 0.78%  
 Прецизійність (precision): 0.76%  
 Повнота (recall): 0.78%  
 F1-score: 0.74%

**Висновок:**

У завданні описується процес створення класифікатора для задачі прогнозування доходу на основі набору даних. Ось основні кроки:

1. **Завантаження даних:** Дані зчитуються з файлу `income_data.txt`, після чого відбираються дві класи:  $\leq 50K$  та  $> 50K$ , кожен з яких обмежується кількістю 25,000 зразків.
2. **Кодування даних:** За допомогою `LabelEncoder` рядкові дані перетворюються на числові для подальшої обробки.
3. **Побудова класифікатора:** Використовується модель `SVM`, зокрема `OneVsOneClassifier` разом з `LinearSVC`, щоб навчити класифікатор на цих даних.
4. **Прогнозування тестової точки:** Тестова точка також кодується, після чого класифікатор передбачає, що вона належить до класу  $\leq 50K$ .

Отже, тестова точка була віднесена до класу  $\leq 50K$  згідно з результатом класифікатора.

**Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами**

```
✓ [14] import numpy as np
0a import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.multiclass import OneVsOneClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

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

```
✓ [3] # Читання даних
0a X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000
```

```
✓ [4] with open(input_file, 'r') as f:
0a     for line in f:
        if count_class1 >= max_datapoints and count_class2 >= max_datapoints:
            break
        if '?' in line:
            continue

        data = line.strip().split(',')

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

```

0a # 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[1]):
    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)

```

```

0a [8] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)
results = {}

```

```

2m [11] # Поліноміальне ядро
poly_classifier = SVC(kernel='poly', degree=3, random_state=0)
poly_classifier.fit(X_train, y_train)
y_pred_poly = poly_classifier.predict(X_test)

```

```

0a [15] results['polynomial kernel'] = {
    'Accuracy': accuracy_score(y_test, y_pred_poly),
    'Precision': precision_score(y_test, y_pred_poly, average='weighted'),
    'Recall': recall_score(y_test, y_pred_poly, average='weighted'),
    'F1-score': f1_score(y_test, y_pred_poly, average='weighted')
}

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use 'zero_division' parameter to control this behavior.
_warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
<

```

```

0a [16] rbf_classifier = SVC(kernel='rbf', random_state=0)
rbf_classifier.fit(X_train, y_train)
y_pred_rbf = rbf_classifier.predict(X_test)

```

```

0a [17] results['rbf kernel'] = {
    'Accuracy': accuracy_score(y_test, y_pred_rbf),
    'Precision': precision_score(y_test, y_pred_rbf, average='weighted'),
    'Recall': recall_score(y_test, y_pred_rbf, average='weighted'),
    'F1-score': f1_score(y_test, y_pred_rbf, average='weighted')
}

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use 'zero_division' parameter to control this behavior.
_warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
<

```

```


0a [18] sigmoid_classifier = SVC(kernel='sigmoid', random_state=0)
sigmoid_classifier.fit(X_train, y_train)
y_pred_sigmoid = sigmoid_classifier.predict(X_test)


```

```

0a [19] results['sigmoid kernel'] = {
    'Accuracy': accuracy_score(y_test, y_pred_sigmoid),
    'Precision': precision_score(y_test, y_pred_sigmoid, average='weighted'),
    'Recall': recall_score(y_test, y_pred_sigmoid, average='weighted'),
    'F1-score': f1_score(y_test, y_pred_sigmoid, average='weighted')
}

```

```
0a ✓  for kernel, metrics in results.items():
    print(f"Results for {kernel}:")
    print(f"Accuracy: {metrics['Accuracy']:.2f}")
    print(f"Precision: {metrics['Precision']:.2f}")
    print(f"Recall: {metrics['Recall']:.2f}")
    print(f"F1-score: {metrics['F1-score']:.2f}\n")
```

 Results for Polynomial Kernel:  
Accuracy: 0.74  
Precision: 0.55  
Recall: 0.74  
F1-score: 0.64

Results for RBF Kernel:  
Accuracy: 0.74  
Precision: 0.55  
Recall: 0.74  
F1-score: 0.64


Results for Sigmoid Kernel:  
Accuracy: 0.61  
Precision: 0.61  
Recall: 0.61  
F1-score: 0.61

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

### **КРОК 1. ЗАВАНТАЖЕННЯ ТА ВИВЧЕННЯ ДАНИХ**







```
✓ 0s  from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
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)
```

```
✓ 0s [3] print(dataset.shape)
```

```
 (150, 5)
```

```
✓ 0s [4] print(dataset.head(20))
```

```
      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
```

```
0s # Статистичні зведення методом describe
print(dataset.describe())
```

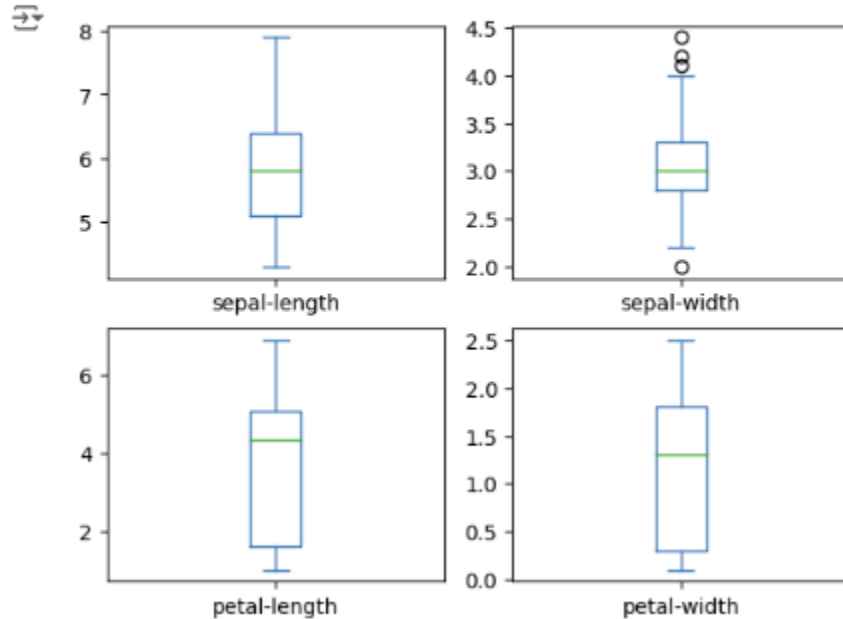
```
↵
      sepal-length  sepal-width  petal-length  petal-width
count    150.000000    150.000000    150.000000    150.000000
mean         5.843333         3.054000         3.758667         1.198667
std          0.828066         0.433594         1.764420         0.763161
min          4.300000         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
```

```
0s # Розподіл за атрибутом class
print(dataset.groupby('class').size())
```

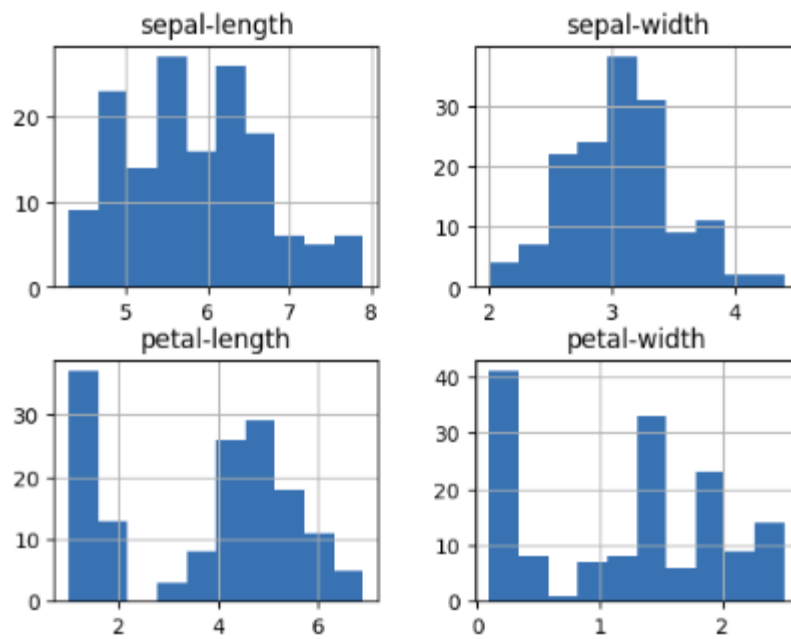
```
↵
class
Iris-setosa      50
Iris-versicolor  50
Iris-virginica   50
dtype: int64
```

## КРОК 2. ВІЗУАЛІЗАЦІЯ ДАНИХ

```
1s # Діаграма розмаху
dataset.plot(kind='box', subplots=True, layout=(2,2),
sharex=False, sharey=False)
pyplot.show()
```

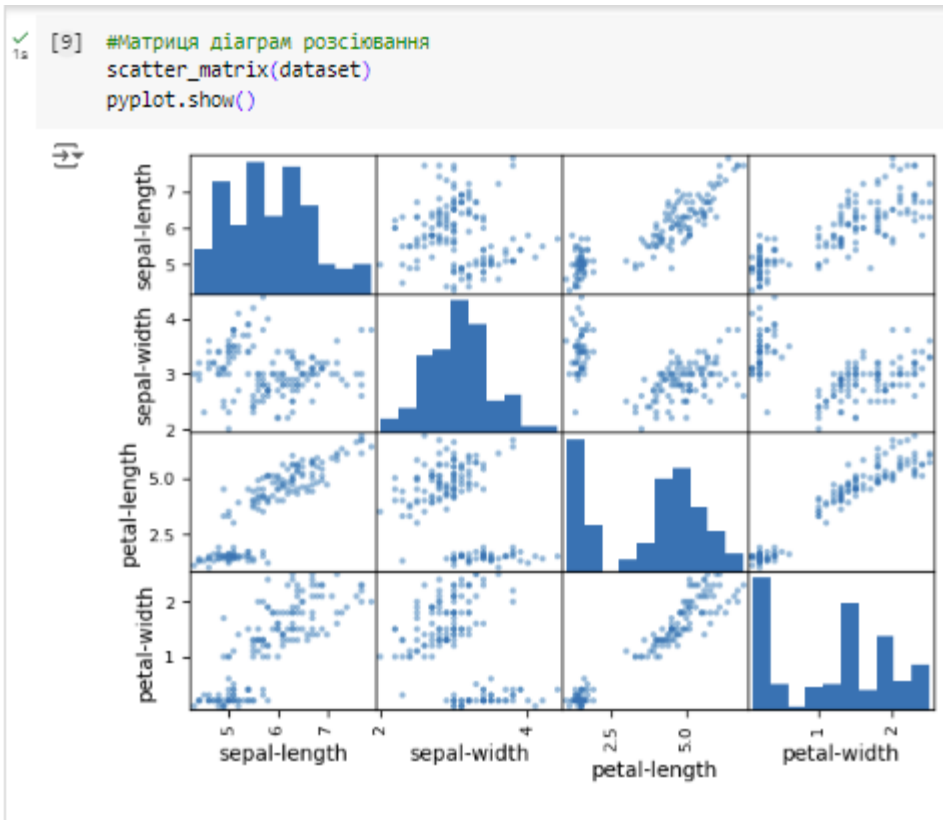


```
✓ 0s # Гістограма розподілу атрибутів датасета  
dataset.hist()  
pyplot.show()
```



```
✓ 1s 9 #Матриця діаграм розсіювання  
scatter_matrix(dataset)  
pyplot.show()
```





### КРОК 3. СТВОРЕННЯ НАВЧАЛЬНОГО ТА ТЕСТОВОГО НАБОРІВ

#### КРОК 3. СТВОРЕННЯ НАВЧАЛЬНОГО ТА ТЕСТОВОГО НАБОРІВ

```
# 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)
```

### КРОК 4. КЛАСИФІКАЦІЯ (ПОБУДОВА МОДЕЛІ)

#### КРОК 4. КЛАСИФІКАЦІЯ (ПОБУДОВА МОДЕЛІ)

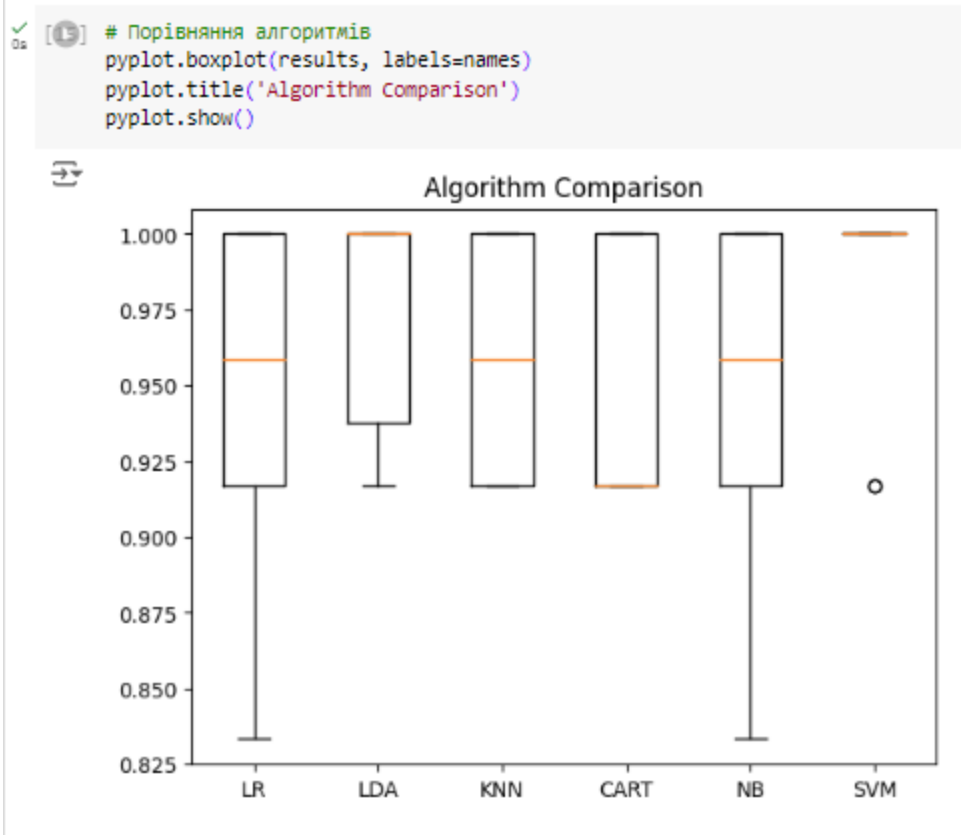
```
[12] # Завантаження алгоритмів моделі
models = []
models.append(('LR', LogisticRegression(solver='liblinear',
multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto'))))

# Оцінюємо модель на кожній ітерації
results = []
names = []

for name, model in models:
    kfold = StratifiedFold(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()))

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. Use OneVsRestClassifier(LogisticRegression(...)) instead. Leave it to its default value to avoid this warning
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. Use OneVsRestClassifier(LogisticRegression(...)) instead. Leave it to its default value to avoid this warning
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. Use OneVsRestClassifier(LogisticRegression(...)) instead. Leave it to its default value to avoid this warning
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. Use OneVsRestClassifier(LogisticRegression(...)) instead. Leave it to its default value to avoid this warning
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. Use OneVsRestClassifier(LogisticRegression(...)) instead. Leave it to its default value to avoid this warning
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. Use OneVsRestClassifier(LogisticRegression(...)) instead. Leave it to its default value to avoid this warning
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. Use OneVsRestClassifier(LogisticRegression(...)) instead. Leave it to its default value to avoid this warning
warnings.warn(
LR: 0.941667 (0.069085)
LDA: 0.975000 (0.033188)
KNN: 0.953333 (0.041667)
CART: 0.950000 (0.040125)
NB: 0.950000 (0.055277)
SVM: 0.993333 (0.033333)
```



#### КРОК 6. ОТРИМАННЯ ПРОГНОЗУ (ПЕРЕДБАЧЕННЯ НА ТРЕНУВАЛЬНОМУ НАБОРІ)

## КРОК 6. ОПТИМІЗАЦІЯ ПАРАМЕТРІВ МОДЕЛІ

```
✓ 0s # Створюємо прогноз на контрольній вибірці
model = SVC(gamma='auto')
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
```

## КРОК 7. ОЦІНКА ЯКОСТІ МОДЕЛІ

### КРОК 7. ОЦІНКА ЯКОСТІ МОДЕЛІ

```
✓ 0s [17] # Оцінюємо прогноз
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

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

## КРОК 8. ОТРИМАННЯ ПРОГНОЗУ (ЗАСТОСУВАННЯ МОДЕЛІ ДЛЯ ПЕРЕДБАЧЕННЯ)

### КРОК 8. ОТРИМАННЯ ПРОГНОЗУ (ЗАСТОСУВАННЯ МОДЕЛІ ДЛЯ ПЕРЕДБАЧЕННЯ)

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# Завантажуємо набір даних Iris
iris_dataset = load_iris()

# Розділяємо дані на навчальні та тестові набори
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)

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

# Створюємо масив із новими даними
X_new = np.array([[5, 2.9, 1, 0.2]])

# Виводимо форму масиву
print("Форма масиву X_new: {}".format(X_new.shape))

# Отримуємо прогноз
prediction = knn.predict(X_new)

# Виводимо результат
print("Прогноз: {}".format(prediction))
print("Спрогнозована мітка: {}".format(iris_dataset['target_names'][prediction]))
```

```
Форма масиву X_new: (1, 4)
Прогноз: [0]
Спрогнозована мітка: ['setosa']
```

Код із оцінкою моделі

```
[22] # Оцінка точності моделі на тестовому наборі
y_pred_test = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_test)

# Виводимо точність моделі
print("Точність моделі на тестовому наборі: {:.2f}%".format(accuracy * 100))
```

```
Точність моделі на тестовому наборі: 97.37%
```

Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання



```

✓ [17] import pandas as pd
1a from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
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, classification_report

# Завантажуємо дані
columns = ['age', 'workclass', 'fnlwtg', 'education', 'education-num', 'marital-status',
           'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
           'hours-per-week', 'native-country', 'income']
data = pd.read_csv('income_data.txt', header=None, names=columns)

# Перетворення цільової змінної 'income' на числові значення
label_encoder = LabelEncoder()
data['income'] = label_encoder.fit_transform(data['income']) # <=50K -> 0, >50K -> 1

✓ [18] categorical_columns = ['workclass', 'education', 'marital-status', 'occupation', 'relationship',
0a 'race', 'sex', 'native-country']
data = pd.get_dummies(data, columns=categorical_columns)

✓ [19] X = data.drop('income', axis=1)
0a y = data['income']

# Розбиваємо дані на навчальні та тестові набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Масштабуємо дані
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

```

✓  
1m



```
models = {
    'Logistic Regression (LR)': LogisticRegression(),
    'Linear Discriminant Analysis (LDA)': LinearDiscriminantAnalysis(),
    'K-Nearest Neighbors (KNN)': KNeighborsClassifier(),
    'Classification and Regression Tree (CART)': DecisionTreeClassifier(),
    'Naive Bayes (NB)': GaussianNB(),
    'Support Vector Machine (SVM)': SVC()
}

# Порівняння моделей
for name, model in models.items():
    # Навчання моделі
    model.fit(X_train_scaled, y_train)

    # Прогнозування
    y_pred = model.predict(X_test_scaled)

    # Оцінка точності
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} - Accuracy: {accuracy:.4f}")
    print(f"Classification Report for {name}:\n{classification_report(y_test, y_pred)}\n")
```



Logistic Regression (LR) - Accuracy: 0.8549  
Classification Report for Logistic Regression (LR):

	precision	recall	f1-score	support
0	0.88	0.93	0.91	7455
1	0.74	0.61	0.66	2314
accuracy			0.85	9769
macro avg	0.81	0.77	0.79	9769
weighted avg	0.85	0.85	0.85	9769

0	0.88	0.93	0.91	7455
1	0.74	0.61	0.66	2314

accuracy			0.85	9769
macro avg	0.81	0.77	0.79	9769
weighted avg	0.85	0.85	0.85	9769

Linear Discriminant Analysis (LDA) - Accuracy: 0.8432  
Classification Report for Linear Discriminant Analysis (LDA):

	precision	recall	f1-score	support
0	0.87	0.93	0.90	7455
1	0.71	0.57	0.63	2314
accuracy			0.84	9769
macro avg	0.79	0.75	0.77	9769
weighted avg	0.84	0.84	0.84	9769

0	0.87	0.93	0.90	7455
1	0.71	0.57	0.63	2314

accuracy			0.84	9769
macro avg	0.79	0.75	0.77	9769
weighted avg	0.84	0.84	0.84	9769

K-Nearest Neighbors (KNN) - Accuracy: 0.8211  
Classification Report for K-Nearest Neighbors (KNN):

	precision	recall	f1-score	support
0	0.87	0.90	0.88	7455
1	0.64	0.56	0.60	2314
accuracy			0.82	9769
macro avg	0.75	0.73	0.74	9769
weighted avg	0.81	0.82	0.82	9769

0	0.87	0.90	0.88	7455
1	0.64	0.56	0.60	2314

accuracy			0.82	9769
macro avg	0.75	0.73	0.74	9769
weighted avg	0.81	0.82	0.82	9769

Classification and Regression Tree (CART) - Accuracy: 0.8097  
Classification Report for Classification and Regression Tree (CART):

```

Logistic Regression (LR) - Accuracy: 0.8549
Classification Report for Logistic Regression (LR):
      precision    recall  f1-score   support

     0       0.88       0.93       0.91       7455
     1       0.74       0.61       0.66       2314

 accuracy         0.85         0.85         0.85         9769
  macro avg       0.81       0.77       0.79         9769
 weighted avg     0.85       0.85       0.85         9769


Linear Discriminant Analysis (LDA) - Accuracy: 0.8432
Classification Report for Linear Discriminant Analysis (LDA):
      precision    recall  f1-score   support

     0       0.87       0.93       0.90       7455
     1       0.71       0.57       0.63       2314

 accuracy         0.84         0.84         0.84         9769
  macro avg       0.79       0.75       0.77         9769
 weighted avg     0.84       0.84       0.84         9769


K-Nearest Neighbors (KNN) - Accuracy: 0.8211
Classification Report for K-Nearest Neighbors (KNN):
      precision    recall  f1-score   support

     0       0.87       0.90       0.88       7455
     1       0.64       0.56       0.60       2314

 accuracy         0.82         0.82         0.82         9769
  macro avg       0.75       0.73       0.74         9769
 weighted avg     0.81       0.82       0.82         9769


Classification and Regression Tree (CART) - Accuracy: 0.8097
Classification Report for Classification and Regression Tree (CART):
      precision    recall  f1-score   support

     0       0.88       0.87       0.87       7455
     1       0.59       0.63       0.61       2314

 accuracy         0.81         0.81         0.81         9769
  macro avg       0.74       0.75       0.74         9769
 weighted avg     0.81       0.81       0.81         9769


Naive Bayes (NB) - Accuracy: 0.4123
Classification Report for Naive Bayes (NB):
      precision    recall  f1-score   support

     0       0.97       0.24       0.38       7455
     1       0.28       0.98       0.44       2314

 accuracy         0.41         0.41         0.41         9769
  macro avg       0.63       0.61       0.41         9769
 weighted avg     0.81       0.41       0.39         9769

```

## Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

02 ✓

```
[1] import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
from io import BytesIO

# Завантажуємо набір даних Iris
iris = load_iris()
X, y = iris.data, iris.target

# Розбиваємо дані на навчальні та тестові набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

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

# Прогнозуємо тестові дані
y_pred = clf.predict(X_test)
```

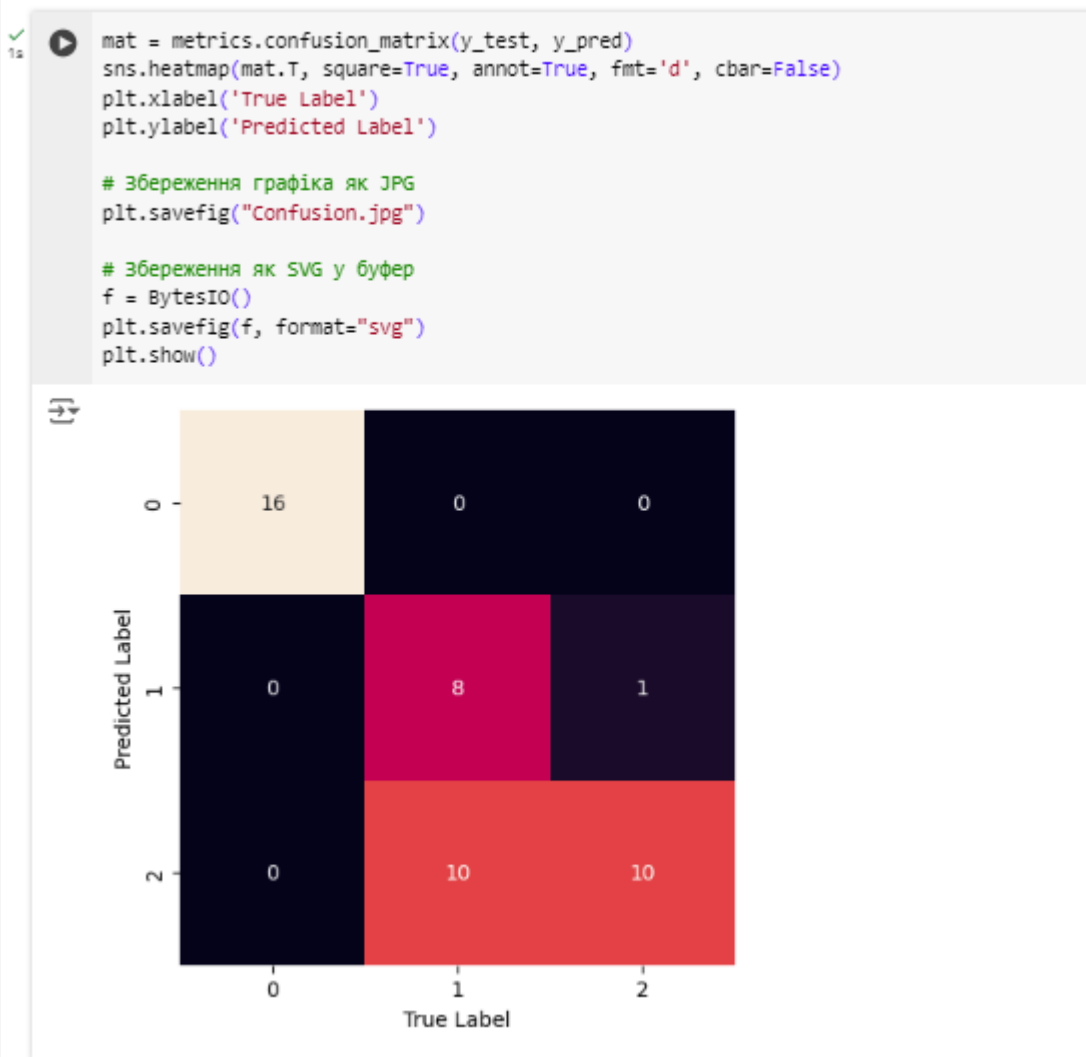
02 ✓

```
[2] 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'), 4))
print('Recall:', np.round(metrics.recall_score(y_test, y_pred, average='weighted'), 4))
print('F1 Score:', np.round(metrics.f1_score(y_test, y_pred, average='weighted'), 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))
```

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



### Пояснення налаштувань класифікатора Ridge

1. **tol=1e-2** — це параметр точності, який визначає мінімальне значення зміни коефіцієнтів моделі під час оптимізації. Чим менше це значення, тим точніше проводиться оптимізація, але це може збільшити час роботи.
2. **solver='sag'** — це метод оптимізації, що використовується для підгонки моделі. SAG (Stochastic Average Gradient) — це швидкий метод для лінійних моделей, який добре працює для великих наборів даних.

### Пояснення показників якості

1. **Accuracy (Точність):** Загальна частка правильно передбачених класів серед усіх тестових прикладів. У нашому випадку це показує загальну якість моделі для трьох класів.
2. **Precision (Точність):** Відсоток правильних позитивних передбачень з усіх передбачених позитивів. Враховує, скільки разів модель правильно передбачила певний клас.
3. **Recall (Повнота):** Відсоток правильних позитивних передбачень з усіх реальних позитивів. Враховує, наскільки добре модель розпізнає позитивні випадки.
4. **F1 Score:** Гармонійне середнє між точністю та повнотою. Важливий показник, коли між точністю і повнотою є компроміс.
5. **Cohen Kappa Score:** Коефіцієнт Коена Каппа вимірює рівень угоди між двома класифікаторами, враховуючи випадкові збіги. Він дає показник від -1 (повна незгода) до 1 (повна згода).
6. **Matthews Corrcoef:** Коефіцієнт кореляції Метьюза (MCC) — це метрика, яка враховує всі чотири значення з матриці плутанини (TP, TN, FP, FN) і дає оцінку від -1 до 1. Це один з найточніших способів оцінки якості класифікатора, особливо при дисбалансі класів.

На зображенні **Confusion.jpg** показано матрицю плутанини (**confusion matrix**), яка використовується для оцінки ефективності класифікаційної моделі. Вона показує, як добре модель класифікує зразки у відповідні класи, порівнюючи передбачені значення з фактичними (істинними) значеннями.

### Матриця плутанини

Матриця плутанини (**confusion matrix**) показує кількість правильних і неправильних передбачень для кожного класу. Кожен рядок представляє

реальні значення, а кожен стовпець — передбачені значення. На графіку ми бачимо:

- Клітини на діагоналі представляють кількість правильно передбачених значень для кожного класу.
- Всі інші клітини — це неправильні передбачення (помилки моделі).

Зображення **Confusion.jpg** відображає точність моделі у вигляді матриці плутанини для кожного класу (три класи Iris).

### **Пояснення коефіцієнтів Коена Каппа та Метьюза**

1. **Коефіцієнт Коена Каппа** вимірює угоду між двома класифікаторами, враховуючи можливість випадкових угод. Значення близьке до 1 означає високу узгодженість, а значення близьке до 0 означає, що угода не краща за випадкову.
2. **Коефіцієнт кореляції Метьюза (MCC)** — це метрика, яка враховує всі можливі типи передбачень (TP, TN, FP, FN). Вона особливо корисна при нерівномірному розподілі класів, оскільки дає збалансовану оцінку якості класифікації. Значення MCC варіюються від -1 (повна невідповідність) до 1 (ідеальна відповідність).

**GitHub** - <https://github.com/TAMOTO24/-Intelligen-Systems>