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.

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

Код з виправленими помилками:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn.model_selection import train_test_split, cross_val_score

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

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

with open(input_file, 'r') as f:
    for line in f.readlines():
        if count_class1 >= max_datapoints and count_class2 >= max_datapoints:
            break

        if '?' in line:
            continue

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

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

if X.size == 0:
    raise ValueError("No data read from the file or all lines contained missing
values ('?')")
```

```
# 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:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label_encoder.append(le)

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

# Розділення 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=5)

# Створення та навчання SVM-класифікатора з параметрами dual=False і збільшеною
кількістю ітерацій
classifier = OneVsOneClassifier(LinearSVC(random_state=0, dual=False,
max_iter=10000))
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)

# Обчислення F-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).reshape(1, -1)

# 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])
```

F міра та результат класифікації:

```
F1 score: 76.12%
<=50K
```

F-міра — це середнє значення між точністю і повнотою класифікатора

Це означає, що середня точність і повнота нашого класифікатора становлять приблизно 76.12%

Інші дані:

Тут наведено ще акуратність, повнота, точність , а також матриця конфузій

Результати, визначені за допомогою використання імпортованих та написання своїх функцій. Видно, що результати майже однакові

```
Accuracy: 79.56%
Recall: 27.90%
Precision: 77.90%
```

```
Accuracy: 79.56%
Recall: 79.56%
Precision: 79.26%
F1 Score: 75.75%
<=50K
{0: 22654, 1: 7508}
[[4370  122]
 [1111  430]]
```

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами

Імпортуємо

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, recall_score, precision_score,
f1_score
```

додаємо код:

```
# Створення SVM з поліноміальним ядром
poly_svm = SVC(kernel='poly', degree=8)
poly_svm.fit(X_train, y_train)
y_pred_poly = poly_svm.predict(X_test)

# Створення SVM з гаусовим ядром
rbf_svm = SVC(kernel='rbf')
```

```
rbf_svm.fit(X_train, y_train)
y_pred_rbf = rbf_svm.predict(X_test)

# Створення SVM з сигмоїдальним ядром
sigmoid_svm = SVC(kernel='sigmoid')
sigmoid_svm.fit(X_train, y_train)
y_pred_sigmoid = sigmoid_svm.predict(X_test)

print("Shape of y_test:", y_test.shape)
print("Shape of y_test_pred:", y_test_pred.shape)

# Перевірка помилок під час побудови моделей SVM
if not hasattr(poly_svm, "fit"):
    raise ValueError("SVM model with polynomial kernel was not properly trained.")
if not hasattr(rbf_svm, "fit"):
    raise ValueError("SVM model with RBF kernel was not properly trained.")
if not hasattr(sigmoid_svm, "fit"):
    raise ValueError("SVM model with sigmoid kernel was not properly trained.")

# Обчислення показників якості класифікації для поліноміального SVM
accuracy_poly = accuracy_score(y_test, y_pred_poly)
precision_poly = precision_score(y_test, y_pred_poly)
recall_poly = recall_score(y_test, y_pred_poly)
f1_poly = f1_score(y_test, y_pred_poly)

# Обчислення показників якості класифікації для гаусового SVM
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
precision_rbf = precision_score(y_test, y_pred_rbf)
recall_rbf = recall_score(y_test, y_pred_rbf)
f1_rbf = f1_score(y_test, y_pred_rbf)

# Обчислення показників якості класифікації для сигмоїдального SVM
accuracy_sigmoid = accuracy_score(y_test, y_pred_sigmoid)
precision_sigmoid = precision_score(y_test, y_pred_sigmoid)
recall_sigmoid = recall_score(y_test, y_pred_sigmoid)
f1_sigmoid = f1_score(y_test, y_pred_sigmoid)


# Виведення результатів

print("\nPolynomial SVM:")
print(f"Accuracy: {accuracy_poly * 100:.2f}%")
print(f"Recall: {recall_poly * 100:.2f}%")
print(f"Precision: {precision_poly * 100:.2f}%")
print(f"F1 Score: {f1_poly * 100:.2f}%")

print("\nGaussian SVM:")
print(f"Accuracy: {accuracy_rbf * 100:.2f}%")
print(f"Recall: {recall_rbf * 100:.2f}%")
print(f"Precision: {precision_rbf * 100:.2f}%")
print(f"F1 Score: {f1_rbf * 100:.2f}%")

print("\nSigmoid SVM:")
print(f"Accuracy: {accuracy_sigmoid * 100:.2f}%")
print(f"Recall: {recall_sigmoid * 100:.2f}%")
print(f"Precision: {precision_sigmoid * 100:.2f}%")
print(f"F1 Score: {f1_sigmoid * 100:.2f}%")
```

Встановлюємо для швидкості

```
max_datapoints = 1000
```

Результат:

```
Polynomial SVM:
Accuracy: 53.75%
Recall: 3.68%
Precision: 77.78%
F1 Score: 7.04%

Gaussian SVM:
Accuracy: 53.25%
Recall: 71.58%
Precision: 50.56%
F1 Score: 59.26%

Sigmoid SVM:
Accuracy: 51.00%
Recall: 48.95%
Precision: 48.44%
F1 Score: 48.69%
```

За результатами тренування видно, що найкраще виконує завдання класифікації модель з ядром гаусового типу (RBF kernel). Це підтверджується найвищим значенням метрик точності, відгукнування, точності і F1-показника серед усіх трьох моделей.

Модель з поліноміальним ядром має найнижчі значення метрик, що може свідчити про недообчисленість моделі або неадекватність вибору параметрів.

Отже, на основі цих результатів можна зробити висновок, що для даного завдання класифікації найбільш ефективним є використання SVM з гаусовим ядром.

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(iris_dataset['data'][:5])
```

Виведіть значення ознак для перших п'яти прикладів:

```
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)
[[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]]
```

```
print("Тип масиву target: {}".format(type(iris_dataset['target'])))
print("Відповіді:\n{}".format(iris_dataset['target']))
```

```
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 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]
```

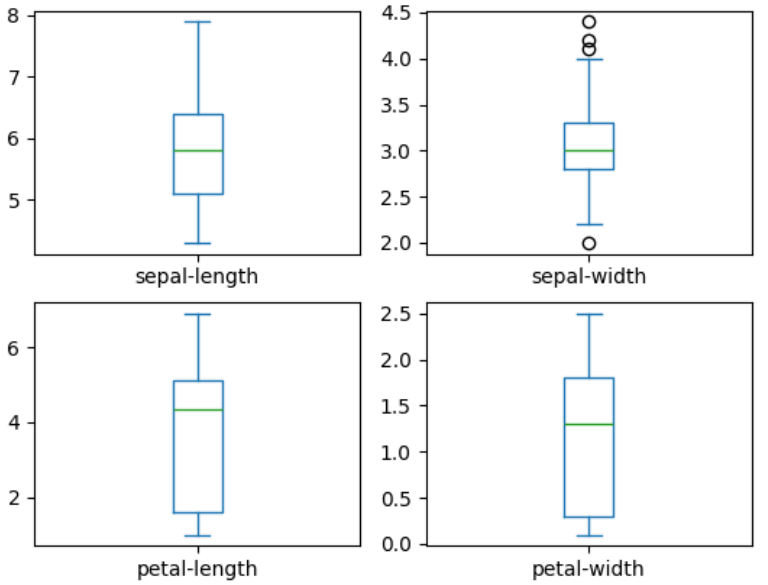
Крок 1

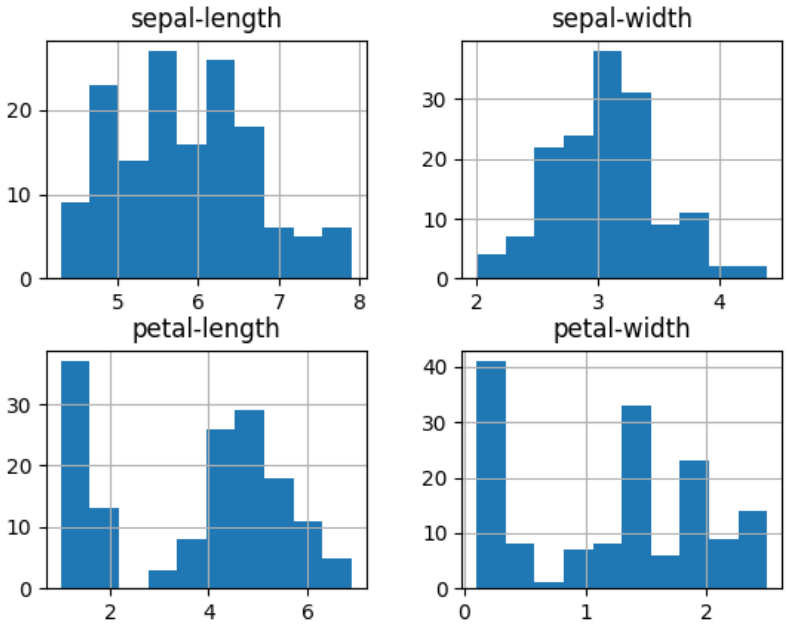
```
# Завантаження датасету
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)

# Аналіз даних
print(dataset.shape)
print(dataset.head(20))
print(dataset.describe())
print(dataset.groupby('class').size())
```

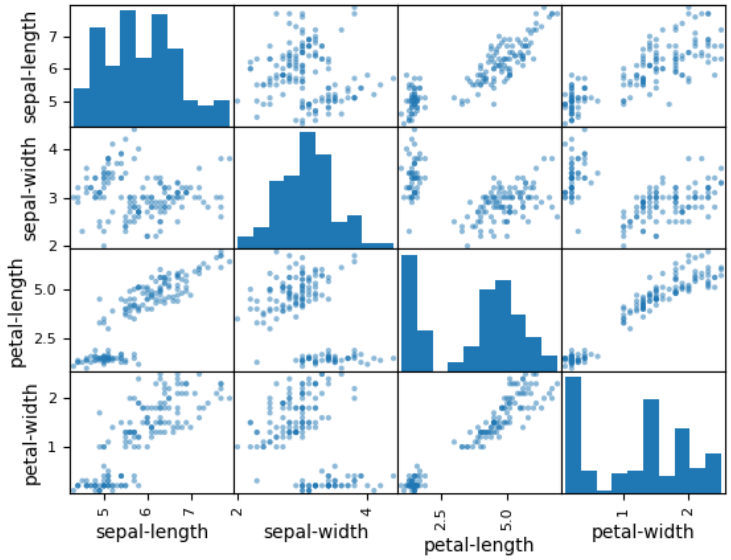
```
(150, 5)
    sepal-length  sepal-width  petal-length  petal-width        class
0            5.1          3.5           1.4          0.2  Iris-setosa
1            4.9          3.0           1.4          0.2  Iris-setosa
2            4.7          3.2           1.3          0.2  Iris-setosa
3      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

Крок 2

```
# 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()
```



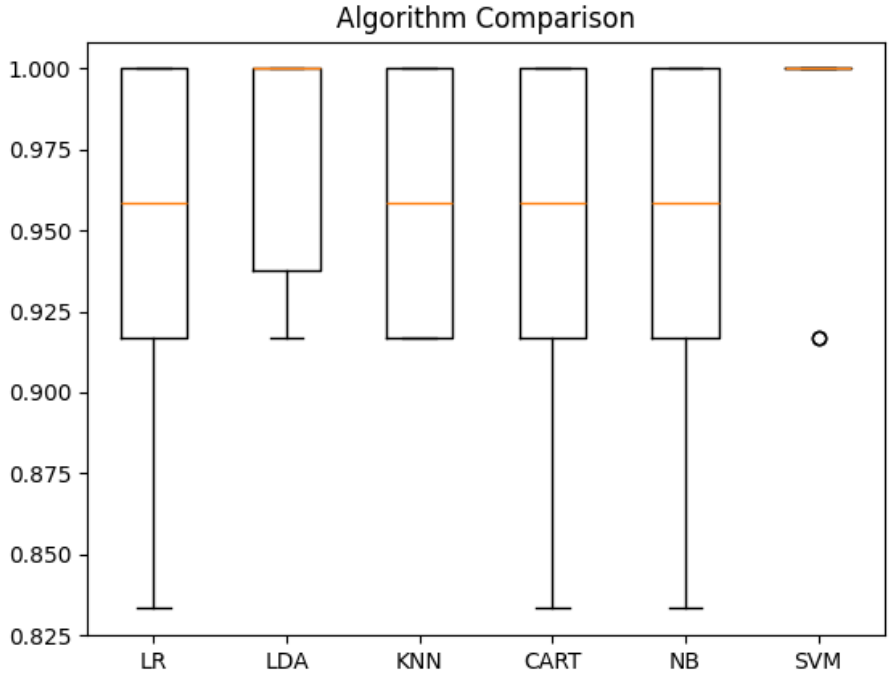




КРОК 3-4

```
# 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()
```



Крок 5

Частково ми опробовували цей крок у попередньому завданні, коли спробували міняти ядро алгоритму SVM.

Крок 6-7

```
# 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))
```

```
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
```

Свій код:

```python
# 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))


# Отримання нових даних
X_new = [[5, 2.9, 1, 0.2]]  # Нові дані про ірис (довжина чашолистка, ширина чашолистка, довжина пелюстки, ширина пелюстки)

# Застосування моделі для прогнозу
prediction = model.predict(X_new)


# Виведення результату прогнозу
print("Прогноз: {}".format(prediction))
```

```
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

Прогноз: ['Iris-setosa']
```

Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1

```
# Розділення 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=42)

models = {
    'Logistic Regression': LogisticRegression(),
    'Linear Discriminant Analysis': LinearDiscriminantAnalysis(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Naive Bayes': GaussianNB(),
    'Support Vector Machine': SVC()
}

# Показники якості класифікації для кожної моделі
for name, model in models.items():
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)
    accuracy = accuracy_score(Y_test, Y_pred)
    report = classification_report(Y_test, Y_pred)
    matrix = confusion_matrix(Y_test, Y_pred)

    print(f"Model: {name}")
    print(f"Accuracy: {accuracy}")
    print("Confusion Matrix:")
    print(matrix)
    print("Classification Report:")
    print(report)
    print("\n")
```

Результати та порівняємо різні алгоритми класифікації для набору даних з файлу income_data.txt.

1 Логістична регресія (LR)

2 Лінійний дискримінантний аналіз (LDA)

3 Метод k-найближчих сусідів (KNN)

4 Класифікація та регресія за допомогою дерев (CART)

5 Наївний баєсовський класифікатор (NB)

6 Метод опорних векторів (SVM)

```
Model: Logistic Regression
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

```
Model: Linear Discriminant Analysis
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

```
Model: K-Nearest Neighbors
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

```
Model: Decision Tree
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

```
Model: Naive Bayes
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

```
Model: Naive Bayes
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

Всі шість моделей показали точність 100% на даному наборі даних. Це може бути наслідком кількості даних та їх якості

Логістична регресія та лінійний дискримінантний аналіз можуть бути більш зручними випадками, коли важлива інтерпретованість результатів. З іншого боку, метод опорних векторів може бути кращим вибором, коли важлива максимальна точність класифікації.

Також важливо врахувати можливість переобучення моделі

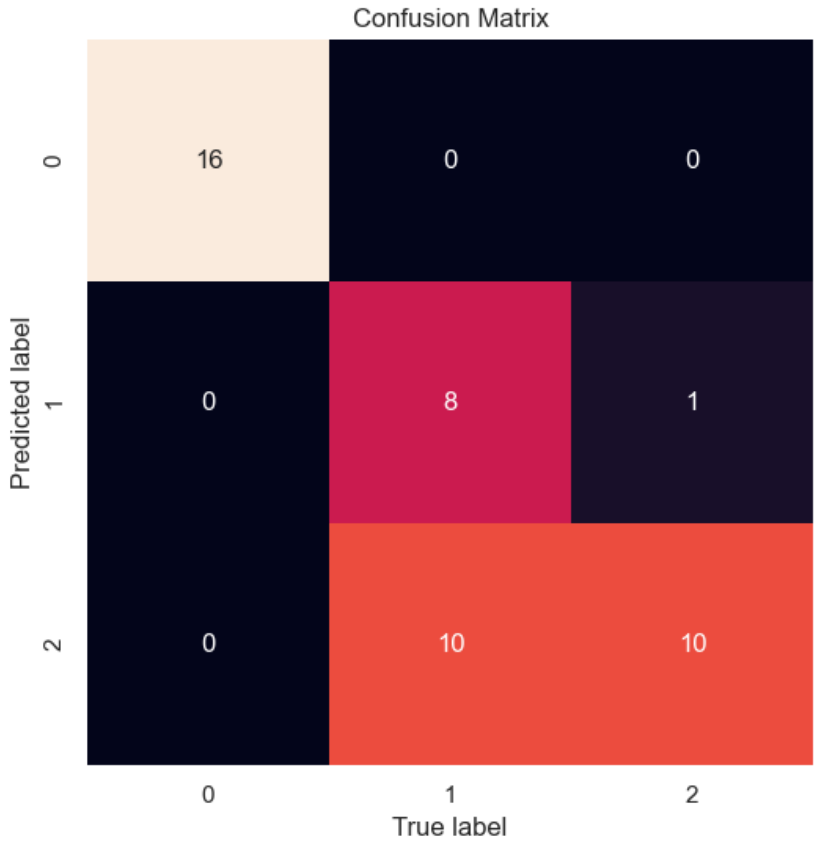
Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

```
C:\Users\ahume>pip install seaborn matplotlib
Requirement already satisfied: seaborn in c:\python311\lib\site-packages (0.13.2)
Requirement already satisfied: matplotlib in c:\python311\lib\site-packages (3.9.0)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\python311\lib\site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=1.2 in c:\python311\lib\site-packages (from seaborn) (2.2.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\python311\lib\site-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in c:\python311\lib\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\python311\lib\site-packages (from matplotlib) (4.49.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\python311\lib\site-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in c:\python311\lib\site-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=8 in c:\python311\lib\site-packages (from matplotlib) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\python311\lib\site-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in c:\python311\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\python311\lib\site-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\python311\lib\site-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: six>=1.5 in c:\python311\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.
0)

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```python
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
from sklearn.metrics import confusion_matrix

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

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

# Прогнозування на тестовій вибірці
y_pred = clf.predict(X_test)

# Розрахунок показників якості класифікації
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))
```

```
# Створення матриці плутанини
mat = confusion_matrix(y_test, y_pred)

# Візуалізація матриці плутанини
sns.set()
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 матриці плутанини у форматі SVG
f = BytesIO()
plt.savefig(f, format="svg")
plt.show()
```

Матриця плутанини



```
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
```

У цьому прикладі для класифікації даних Iris використовується лінійний класифікатор Ridge з наступними налаштуваннями:

tol=1e-2: Значення толерантності для припинення алгоритму. Це визначає, наскільки точним повинен бути розв'язок, щоб алгоритм припинив роботу. Чим менше значення, тим точнішим буде розв'язок, але це може збільшити час роботи алгоритму.

solver="sag": Вказує на використання стохастичного середньоградієнтного методу (Stochastic Average Gradient Descent) для оптимізації. Цей метод є ефективним для великих наборів даних.

Висновок: Під час виконання лабораторної роботи, я за допомогою коду, використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив попередню обробку та класифікацію даних