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

Опис набору вхідних даних:

1. Age (вік): 17 – 90 років
2. Workclass (зайнятість): Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked
3. Fnlwgt (Final weight, ваговий коефіцієнт): числове значення
4. Education (освіта): Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc- acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool
5. Education-num (числове преставлення рівня освіти): числове значення
6. Marital-status (сімейний стан): Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse
7. Occupation (професія): Tech-support, Craft-repair, Other-service, Sales, Exec- managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed- Forces
8. Relationship (відносити): Wife, Own-child, Husband, Not-in-family, Other- relative, Unmarried
9. Race (раса): White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
10. Sex (стать): Female, Male
11. Capital-gain (капіталовигода): числове значення
12. Capital-loss (капіталовтрата): числове значення
13. Hours-per-week (години на тиждень): числове значення
14. Native-country (Батьківщина): United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand- Netherlands.
15. Клас доходу: >=50K чи <= 50K.

```
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

# Input file containing data
input_file = 'income_data.txt'

# Read the data
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

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

# Convert to numpy array
X = np.array(X)

# Convert string data to numerical data
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)

scaller = preprocessing.MinMaxScaler(feature_range=(0, 1))
X = scaller.fit_transform(X)

# Create SVM classifier
classifier = OneVsOneClassifier(LinearSVC(random_state=0))

# Train the classifier
classifier.fit(X=X, y=y)

# Cross validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)
classifier = OneVsOneClassifier(LinearSVC(random_state=0))

scaller = preprocessing.MinMaxScaler(feature_range=(0, 1))
X_train = scaller.fit_transform(X_train)


classifier.fit(X=X_train, y=y_train)
y_test_pred = classifier.predict(X_test)

# Compute the F1 score of the SVM classifier
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy', cv=3)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
precision_values = cross_val_score(classifier, X, y, scoring='precision_weighted',
cv=3)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
recall_values = cross_val_score(classifier, X, y, scoring='recall_weighted', cv=3)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")
f1_values = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")

print("F1 score: " + str(round(100*f1.mean(), 2)) + "%")

# Predict output for a test datapoint
input_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married',
'Handlers-cleaners', 'Not-in-family', 'White', 'Male', '0', '0', '40', 'United-
States']

# Encode test datapoint
input_data_encoded = np.array([-1] * len(input_data))
count = 0
for i, item in enumerate(input_data):
    if item.isdigit():
        input_data_encoded[i] = item
    else:
        input_data_encoded[i] = int(label_encoder[count].transform([item]))
        count += 1
input_data_encoded = input_data_encoded.astype(int)
input_data_encoded = [input_data_encoded]
```

```
# Run classifier on encoded datapoint and print output
predicted_class = classifier.predict(input_data_encoded)
print(label_encoder[-1].inverse_transform(predicted_class)[0])
```

```
Accuracy: 82.01%
Precision: 80.96%
Recall: 82.01%
F1: 80.1%
F1 score: 80.1%
>50K
```

Рис.1 Результат виконання

Завдання 2.2

```
Accuracy: 83.5%
Precision: 82.84%
Recall: 83.5%
F1: 83.01%
F1 score: 83.01%
<=50K
```

Рис. 2 Результат виконання (poly ядро)

```
Accuracy: 58.2%
Precision: 57.85%
Recall: 58.2%
F1: 58.02%
F1 score: 58.02%
<=50K
```

Рис.3 Результат виконання (sigmoid ядро)

У результаті визначено, що ядро RBF продемонструвало добрі результати, маючи невеликий недолік порівняно з поліноміальним ядром, але виграючи у швидкості. З іншого боку, сигмоїдне ядро показало менший результат. З урахуванням цього висновку можна стверджувати, що для нашого конкретного випадку оптимальним вибором буде використання RBF-ядра, яке комбінує високу точність і ефективність.

Завдання 2.3

```
from sklearn.datasets import load_iris
import numpy as np
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

iris_dataset = load_iris()
print(f'Ключі iris_dataset: {iris_dataset.keys()}')
print(iris_dataset['DESCR'][:193] + "\n....")
print(f"Назви відповідей: {iris_dataset['target_names']}")
print(f"Назва ознак: {iris_dataset['feature_names']}")
print(f"Тип масиву data: {type(iris_dataset['data'])}")
print(f"Форма масиву data: {iris_dataset['data'].shape}")
print("Відповіді:\n{}".format(iris_dataset['target']))

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)

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')))

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))

X_new = np.array([[5, 2.9, 1, 0.2]])

for name, model in models:
    model.fit(X_train, Y_train)
    prediction = model.predict(X_new)
    print("Прогноз: {}".format(prediction))
    print(accuracy_score(Y_validation, predictions))
    print(confusion_matrix(Y_validation, predictions))
    print(classification_report(Y_validation, predictions))
```

```
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)
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 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 2 2 2
 2 2]
    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
```

Рис.4 Результат виконання

```
       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
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
```

Рис.5 Результат виконання

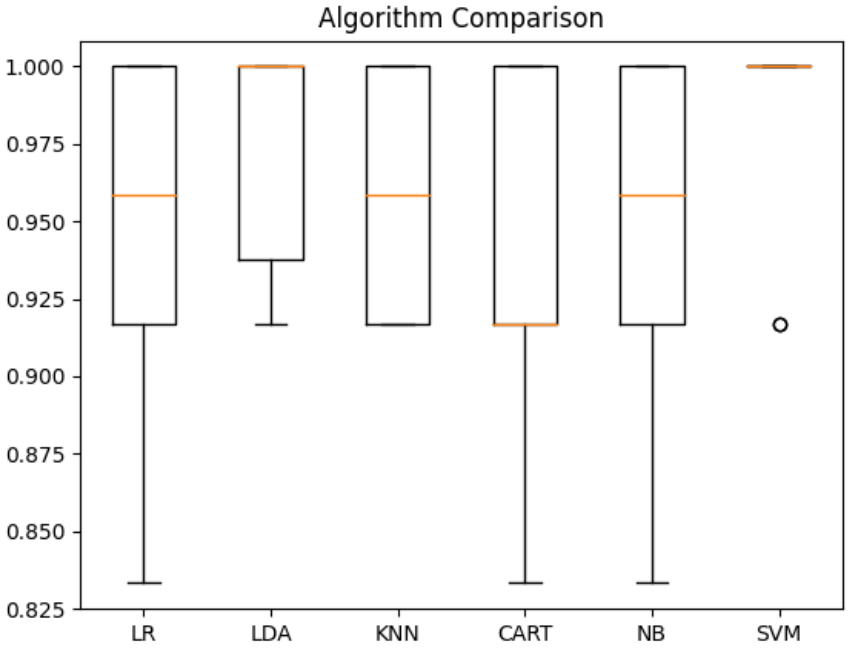

Рис. 6 Рисунок порівняння алгоритмів

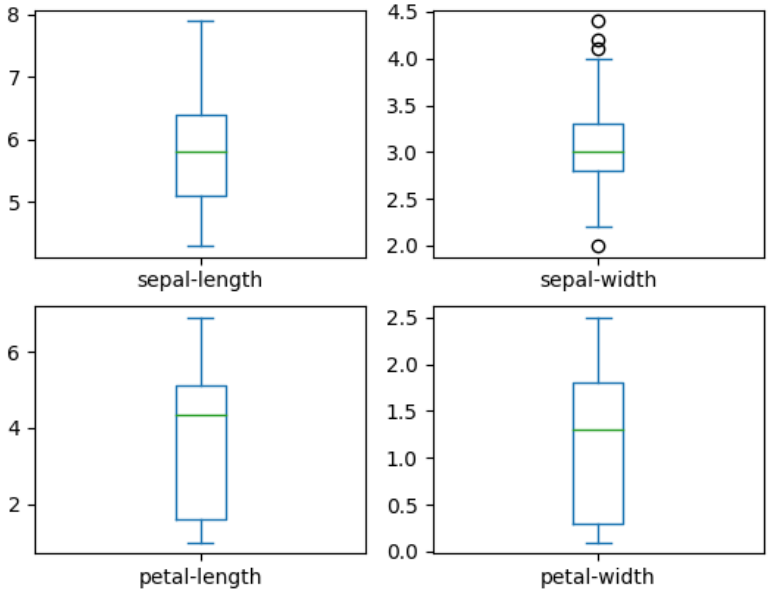

Рис. 7 Результат діаграми розмаху

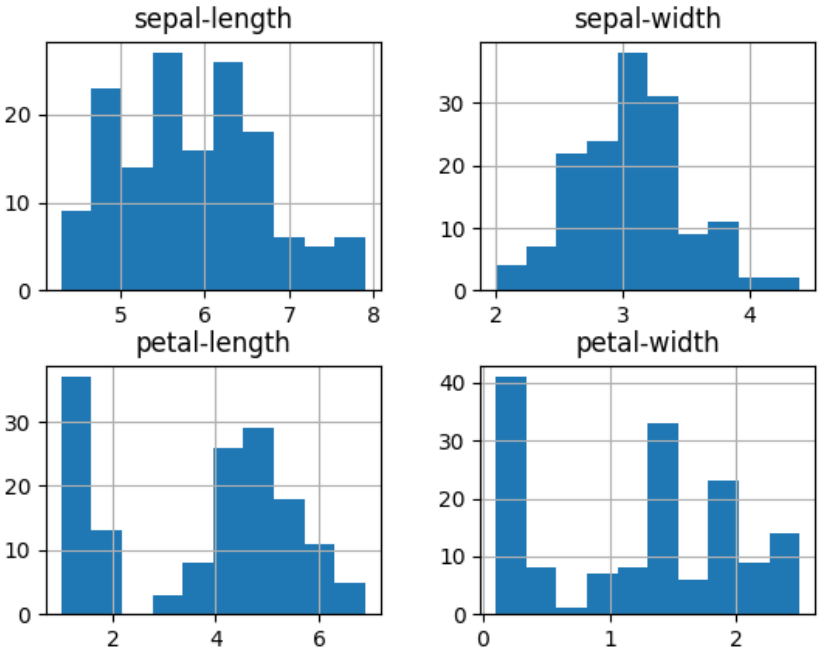

Рис. 8 Гістрограма розподілу атрибутів

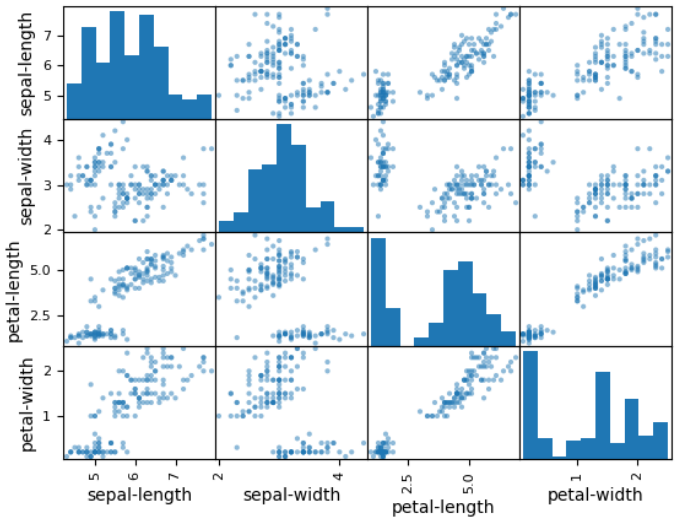

Рис. 9 Матриця діаграми розсіювання

Квітка належала до класу Iris-setosa

Узагальнюючи з діаграм, можемо визначити, що найбільш ефективною виявилася модель лінійного дискримінантного аналізу, хоча вона має певні недоліки, такі як нестійкість та більший час виконання під час тестування.

Завдання 2.4

```
Accuracy: 81.85%
Precision: 80.68%
Recall: 81.85%
F1: 80.13%
F1 score: 80.13%
>50K
```

Рис.10 Точність класифікатора LR

```
Accuracy: 81.35%
Precision: 80.04%
Recall: 81.35%
F1: 79.51%
F1 score: 79.51%
>50K
```

Рис. 11 Точність класифікатора LDA

```
Accuracy: 82.43%
Precision: 81.79%
Recall: 82.43%
F1: 82.01%
F1 score: 82.01%
<=50K
```

Рис. 12 Точність класифікатора KNN

```
Accuracy: 80.49%
Precision: 80.91%
Recall: 80.66%
F1: 80.83%
F1 score: 80.96%
<=50K
```

Рис. 13 Точність класифікатора CART

```
Accuracy: 82.3%
Precision: 81.47%
Recall: 82.3%
F1: 80.26%
F1 score: 80.26%
<=50K
```

Рис. 14 Точність класифікатора SVM

```
Accuracy: 80.1%
Precision: 78.51%
Recall: 80.1%
F1: 77.53%
F1 score: 77.53%
<=50K
```

Рис. 15 Точність класифікатора NB

Завдання 2.5

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

sns.set()
iris = load_iris()
X, y = iris.data, iris.target
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.3,
random_state=0)
clf = RidgeClassifier(tol=1e-2, solver="sag")
clf.fit(Xtrain, ytrain)
ypred = clf.predict(Xtest)

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

mat = confusion_matrix(ytest, ypred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
```

```
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.savefig("Confusion.jpg")
# Save SVG in a fake file object.
f = BytesIO()
plt.savefig(f, format="svg")
```

```
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       0.44      0.89      0.59         9
           2       0.91      0.50      0.65        20

    accuracy                           0.76        45
   macro avg       0.78      0.80      0.75        45
weighted avg       0.85      0.76      0.76        45
```

Рис.16 Результат виконання

У використаному класифікаторі Ridge встановлені наступні налаштування: tol = 1e-2 та solver = "sag". Параметр tol визначає критерій збіжності оптимізації, де оптимізація припиниться, коли різниця між послідовними ітераціями стане меншою за 1e-2. Збільшення tol може прискорити збіжність, але може вплинути на точність результату. solver = "sag" вказує метод вирішення задачі оптимізації, в даному випадку — Stochastic Average Gradient Descent, який застосовується до логістичної регресії.

та 4. Для оцінки моделі використовуються показники якості, такі як accuracy, precision, recall, f1 score, Cohen Kappa score, Metthews Corrcoef.

a. Accuracy визначає долю правильних класифікацій і становить 0.7556.

b. Precision — співвідношення правильних позитивних класифікацій до всіх класифікацій — 0.8333.

c. Recall — співвідношення правильних позитивних класифікацій до загального числа істинних позитивних екземплярів — 0.7556.

d. F1 score — гармонійне середнє точності та повноти — 0.7503.

e. Cohen Kappa score — статистична міра збігу між класифікатором та реальними даними, з урахуванням випадкового збігу — 0.6431.

f. Matthews Corrcoef — міра якості бінарної класифікації, яка враховує і істинно позитивні, і істинно негативні випадки — значення від -1 до 1, де 1 — ідеальна згода, 0 — випадковий збіг, -1 — повна незгода.

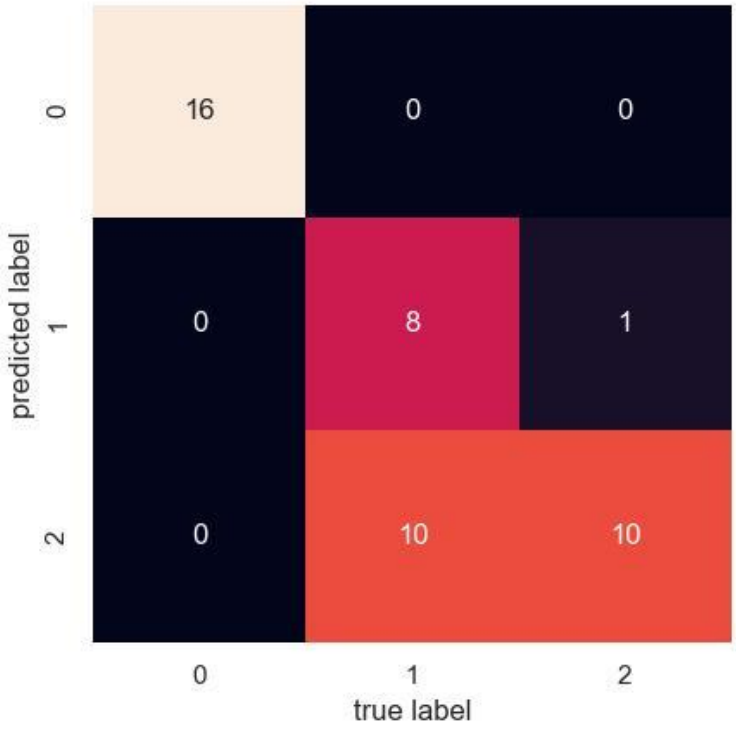

Рис.17 Результат виконання

3. Зображення є карткою матриці помилок, яка відображає кількість правильно класифікованих екземплярів і кількість помилок для кожного класу. За додаванням значень по діагоналі можна визначити загальну кількість правильних класифікацій.

GitHub: https://github.com/unravee1/AI_labs