ЛАБОРАТОРНА РОБОТА № 2

Варіант 17

ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ

**Завдання 1**

Список ознак:

Age – Тип: Числова

Workclass– Тип: Категоріальна

Education – Тип: Категоріальна

Education-num – Тип: Числова

Marital-status – Тип: Категоріальна

Occupation – Тип: Категоріальна

Relationship – Тип: Категоріальна

Race – Тип: Категоріальна

Sex – Тип: Категоріальна

Capital-gain – Тип: Числова

Capital-loss – Тип: Числова

Hours-per-week – Тип: Числова

Native-country – Тип: Категоріальна

Income – Тип: Категоріальна

Лістинг:

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

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

input\_file = 'income\_data.txt'

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.strip().split(', ')

        if data[-1] == '<=50K' and count\_class1 < max\_datapoints:

            X.append(data)

            Y.append(0)

            count\_class1 += 1

        elif data[-1] == '>50K' and count\_class2 < max\_datapoints:

            X.append(data)

            Y.append(1)

            count\_class2 += 1

X = np.array(X)

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 = np.array(Y)

classifier = OneVsOneClassifier(LinearSVC(*random\_state*=0))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, *test\_size*=0.2, *random\_state*=5)

classifier.fit(X\_train, y\_train)

y\_test\_pred = classifier.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_test\_pred)

precision = precision\_score(y\_test, y\_test\_pred, *average*='weighted')

recall = recall\_score(y\_test, y\_test\_pred, *average*='weighted')

f1 = f1\_score(y\_test, y\_test\_pred, *average*='weighted')

print(*f*"Accuracy: {accuracy \* 100*:.2f*}%")

print(*f*"Precision: {precision \* 100*:.2f*}%")

print(*f*"Recall: {recall \* 100*:.2f*}%")

print(*f*"F1 Score: {f1 \* 100*:.2f*}%")

f1\_cv = cross\_val\_score(classifier, X, Y, *scoring*='f1\_weighted', *cv*=3)

print(*f*"F1 score (cross-validation): {f1\_cv.mean() \* 100*:.2f*}%")

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

input\_data\_encoded = np.array([0] \* len(input\_data))

count = 0

for i, item in enumerate(input\_data):

    if item.isdigit():

        input\_data\_encoded[i] = int(item)

    else:

        input\_data\_encoded[i] = label\_encoder[count].transform([item])[0]

        count += 1

input\_data\_encoded = input\_data\_encoded.reshape(1, -1)

predicted\_class = classifier.predict(input\_data\_encoded)

print(*f*"Predicted class: {'<=50K' if predicted\_class[0] == 0 else '>50K'}")

Результат виконання програми:

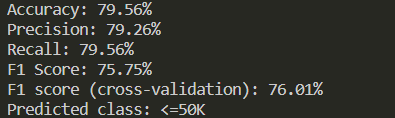
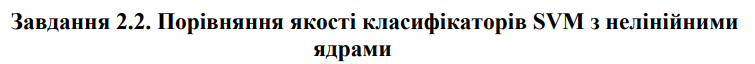


Рис. 1

Тестова точка належить до класу: Income (Дохід)



Завдання було виконано в одному файлі:

import numpy as np

from sklearn import preprocessing

from sklearn.svm import SVC

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.preprocessing import StandardScaler

# Зчитування та попередня обробка даних

input\_file = 'income\_data.txt'

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.strip().split(', ')

        if data[-1] == '<=50K' and count\_class1 < max\_datapoints:

            X.append(data)

            Y.append(0)

            count\_class1 += 1

        elif data[-1] == '>50K' and count\_class2 < max\_datapoints:

            X.append(data)

            Y.append(1)

            count\_class2 += 1

X = np.array(X)

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 = np.array(Y)

scaler = StandardScaler()

X = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, *test\_size*=0.2, *random\_state*=5)

metrics = {}

poly\_svc = SVC(*kernel*='poly', *degree*=2, *random\_state*=0)

poly\_svc.fit(X\_train, y\_train)

y\_test\_pred = poly\_svc.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_test\_pred)

precision = precision\_score(y\_test, y\_test\_pred, *average*='weighted')

recall = recall\_score(y\_test, y\_test\_pred, *average*='weighted')

f1 = f1\_score(y\_test, y\_test\_pred, *average*='weighted')

metrics['Polynomial Kernel'] = {

    "Accuracy": accuracy \* 100,

    "Precision": precision \* 100,

    "Recall": recall \* 100,

    "F1 Score": f1 \* 100

}

rbf\_svc = SVC(*kernel*='rbf', *random\_state*=0)

rbf\_svc.fit(X\_train, y\_train)

y\_test\_pred = rbf\_svc.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_test\_pred)

precision = precision\_score(y\_test, y\_test\_pred, *average*='weighted')

recall = recall\_score(y\_test, y\_test\_pred, *average*='weighted')

f1 = f1\_score(y\_test, y\_test\_pred, *average*='weighted')

metrics['RBF Kernel'] = {

    "Accuracy": accuracy \* 100,

    "Precision": precision \* 100,

    "Recall": recall \* 100,

    "F1 Score": f1 \* 100

}

sigmoid\_svc = SVC(*kernel*='sigmoid', *random\_state*=0)

sigmoid\_svc.fit(X\_train, y\_train)

y\_test\_pred = sigmoid\_svc.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_test\_pred)

precision = precision\_score(y\_test, y\_test\_pred, *average*='weighted')

recall = recall\_score(y\_test, y\_test\_pred, *average*='weighted')

f1 = f1\_score(y\_test, y\_test\_pred, *average*='weighted')

metrics['Sigmoid Kernel'] = {

    "Accuracy": accuracy \* 100,

    "Precision": precision \* 100,

    "Recall": recall \* 100,

    "F1 Score": f1 \* 100

}

for kernel, scores in metrics.items():

    print(*f*"\n{kernel}:")

    print(*f*"Accuracy: {scores['Accuracy']*:.2f*}%")

    print(*f*"Precision: {scores['Precision']*:.2f*}%")

    print(*f*"Recall: {scores['Recall']*:.2f*}%")

    print(*f*"F1 Score: {scores['F1 Score']*:.2f*}%")

Результат виконання програми:

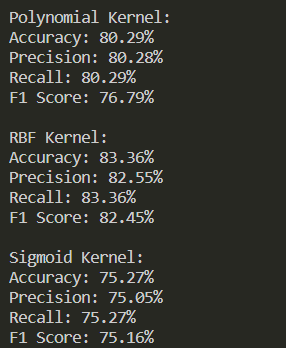


Рис. 2

Краще всіх себе показав RBF, маючи найкращі показники серед усіх.



Тестовий код:

from sklearn.datasets import load\_iris

iris\_dataset = load\_iris()

print("Ключі iris\_dataset: \n{}".format(iris\_dataset.keys()))

print("Опис набору даних:\n{}".format(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 п'яти прикладів:\n{}".format(iris\_dataset['data'][:5]))

print("Тип масиву target:{}".format(type(iris\_dataset['target'])))

print("Відповіді:\n{}".format(iris\_dataset['target']))

Результат виконання програми:

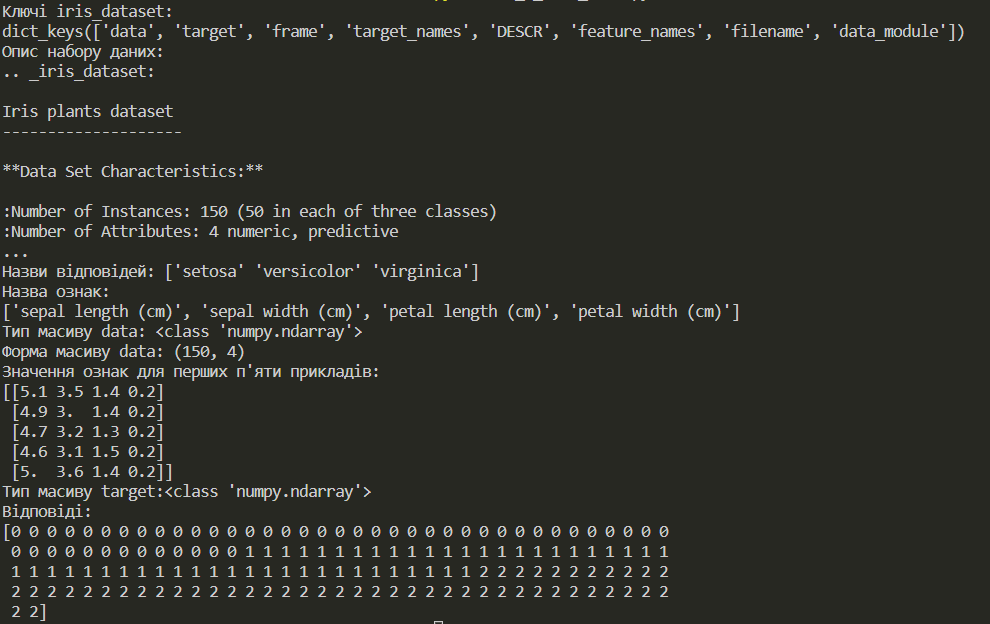


Рис. 3

Код для візуалізації (деяки фрагменти кода з л.р. були винесені в окремі функцій):

from pandas import read\_csv

from pandas.plotting import scatter\_matrix

from matplotlib import pyplot

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)

*def* print\_array\_data():

    print(dataset.shape)

    print(dataset.head(20))

    print(dataset.describe())

    print(dataset.groupby('class').size())

*def* show\_array\_data\_in\_chart():

    dataset.plot(*kind*='box', *subplots*=True, *layout*=(2,2), *sharex*=False, *sharey*=False)

    pyplot.show()

    dataset.hist()

    pyplot.show()

    scatter\_matrix(dataset)

    pyplot.show()

show\_array\_data\_in\_chart()

Результат виконання програми:

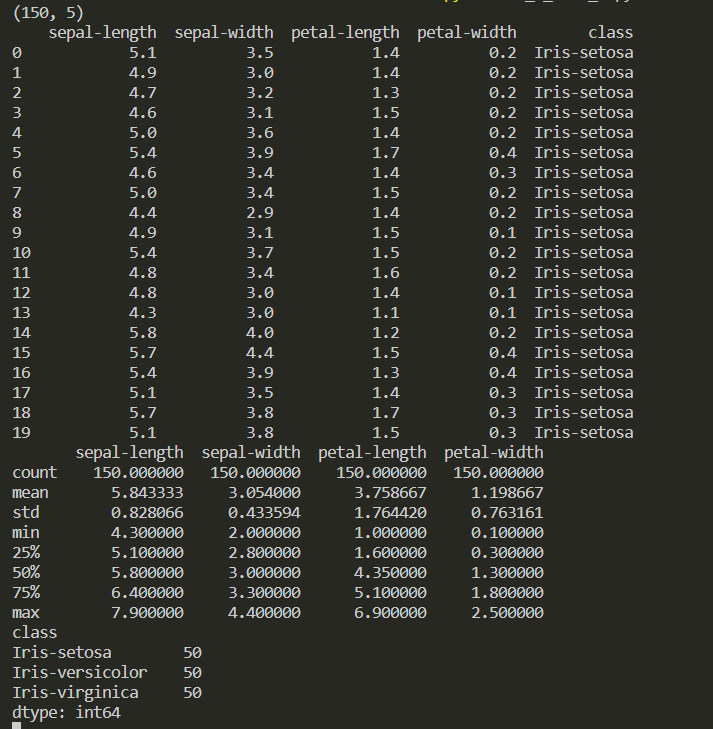


Рис. 4

Результат виконання програми:

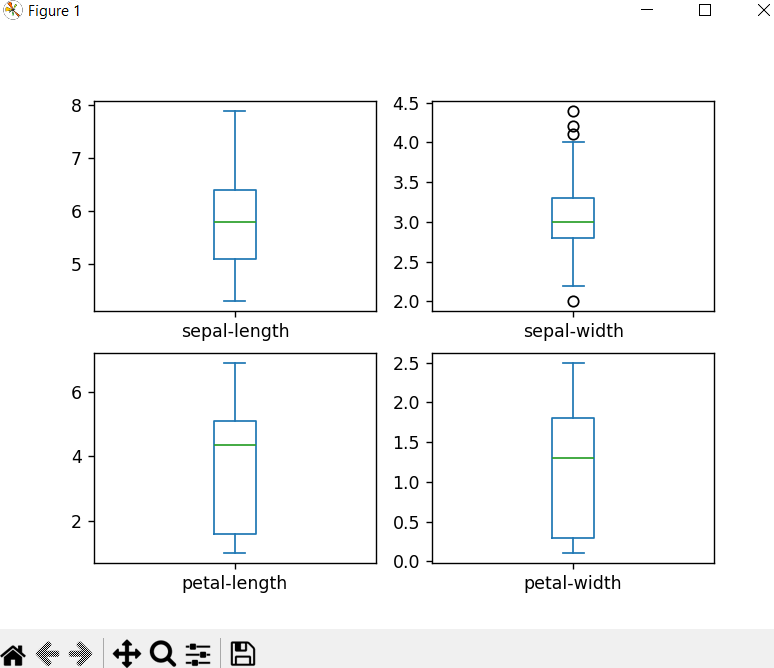


Рис. 5

Результат виконання програми:

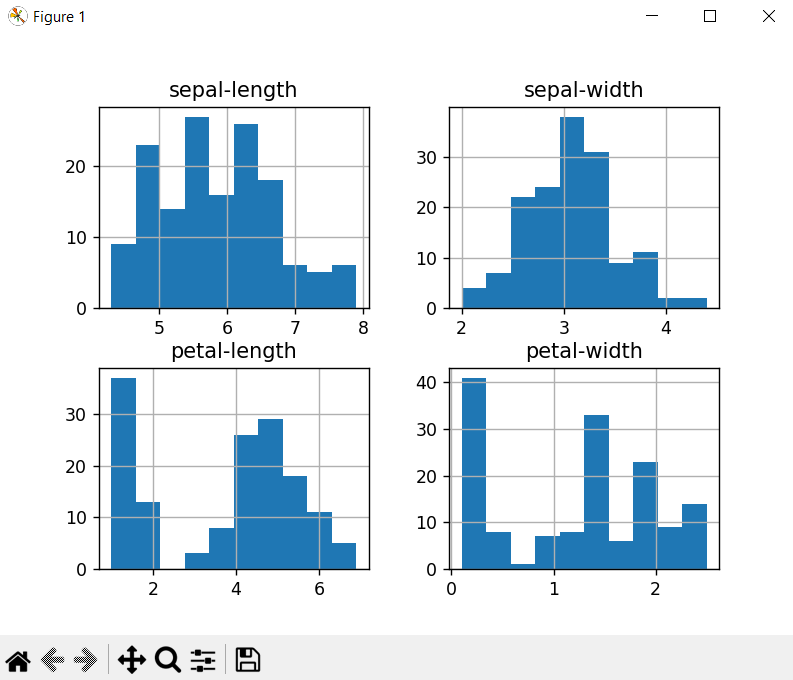


Рис. 6

Результат виконання програми:

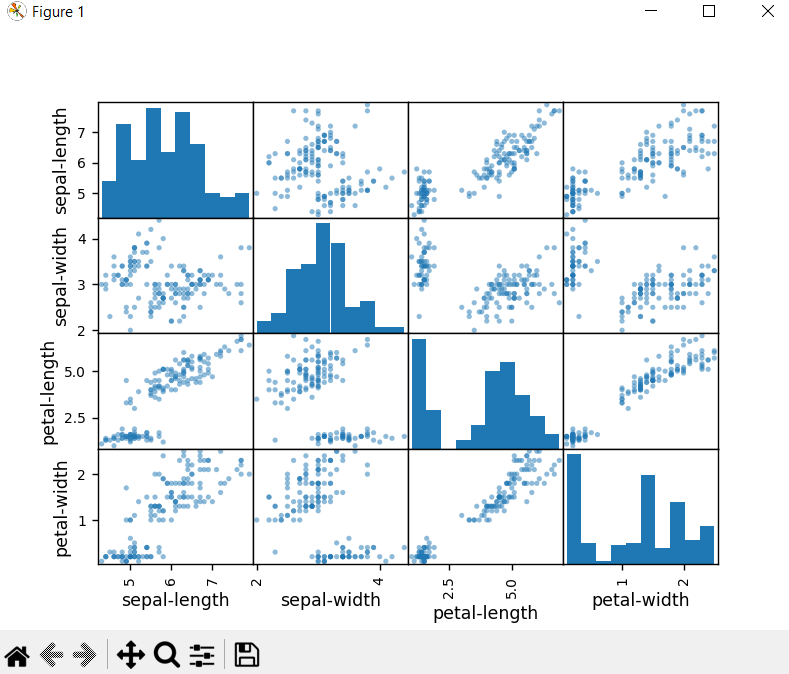


Рис. 7

Вибір та порівняння підходящої моделі

Результат виконання програми:

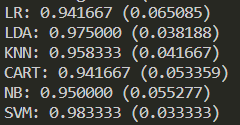


Рис. 8

Результат виконання програми:

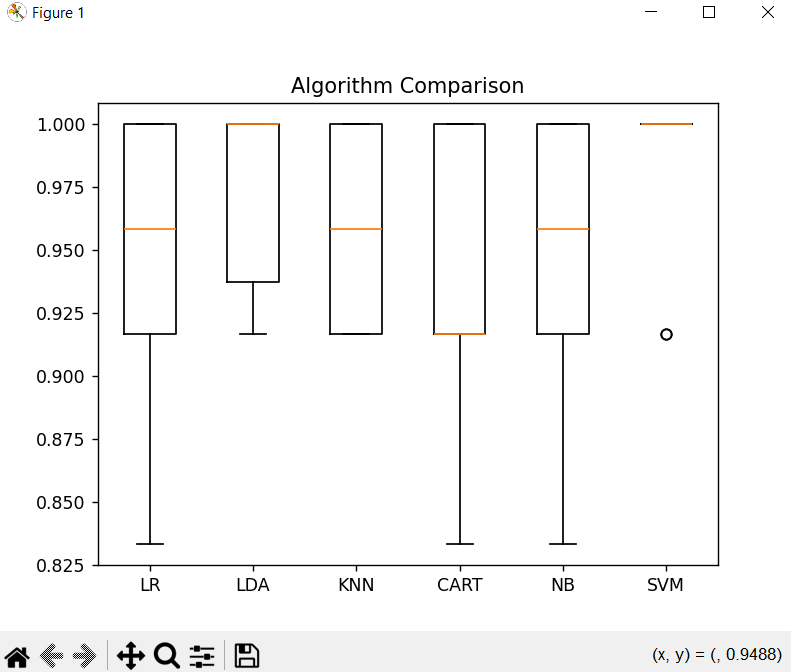


Рис. 9

SVM має найбільшу точність та найменше відхилення серед решти, тому вважаю що варто обрати його.

Оцінка якості:

Результат виконання програми:

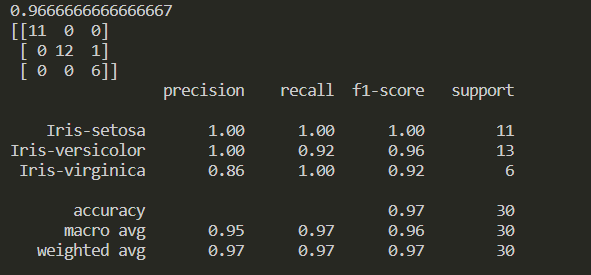


Рис. 10

Також виявлено до якого класу належить квітка з кроку 8:

Результат виконання програми:

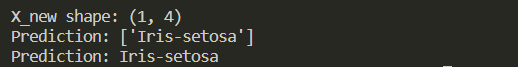


Рис. 11

Лістинг:

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

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)

*def* print\_array\_data():

    print(dataset.shape)

    print(dataset.head(20))

    print(dataset.describe())

    print(dataset.groupby('class').size())

*def* show\_array\_data\_in\_chart():

    dataset.plot(*kind*='box', *subplots*=True, *layout*=(2,2), *sharex*=False, *sharey*=False)

    pyplot.show()

    dataset.hist()

    pyplot.show()

    scatter\_matrix(dataset)

    pyplot.show()

#print\_array\_data()

#show\_array\_data\_in\_chart()

array = dataset.values

X = array[:,0:4]

y = array[:,4]

X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, y, *test\_size*=0.20, *random\_state*=1)

*def* models\_test():

    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()))

    pyplot.boxplot(results, *labels*=names)

    pyplot.title('Algorithm Comparison')

    pyplot.show()

#models\_test()

model = SVC(*gamma*='auto')

model.fit(X\_train, Y\_train)

predictions = model.predict(X\_validation)

print(accuracy\_score(Y\_validation, predictions))

print(confusion\_matrix(Y\_validation, predictions))

print(classification\_report(Y\_validation, predictions))

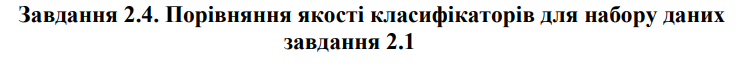
X\_new = np.array([[5, 2.9, 1, 0.2]])

print("X\_new shape: {}".format(X\_new.shape))

prediction = model.predict(X\_new)

print("Prediction: {}".format(prediction))

print("Prediction: {}".format(prediction[0]))



Лістинг:

Результат виконання програми:

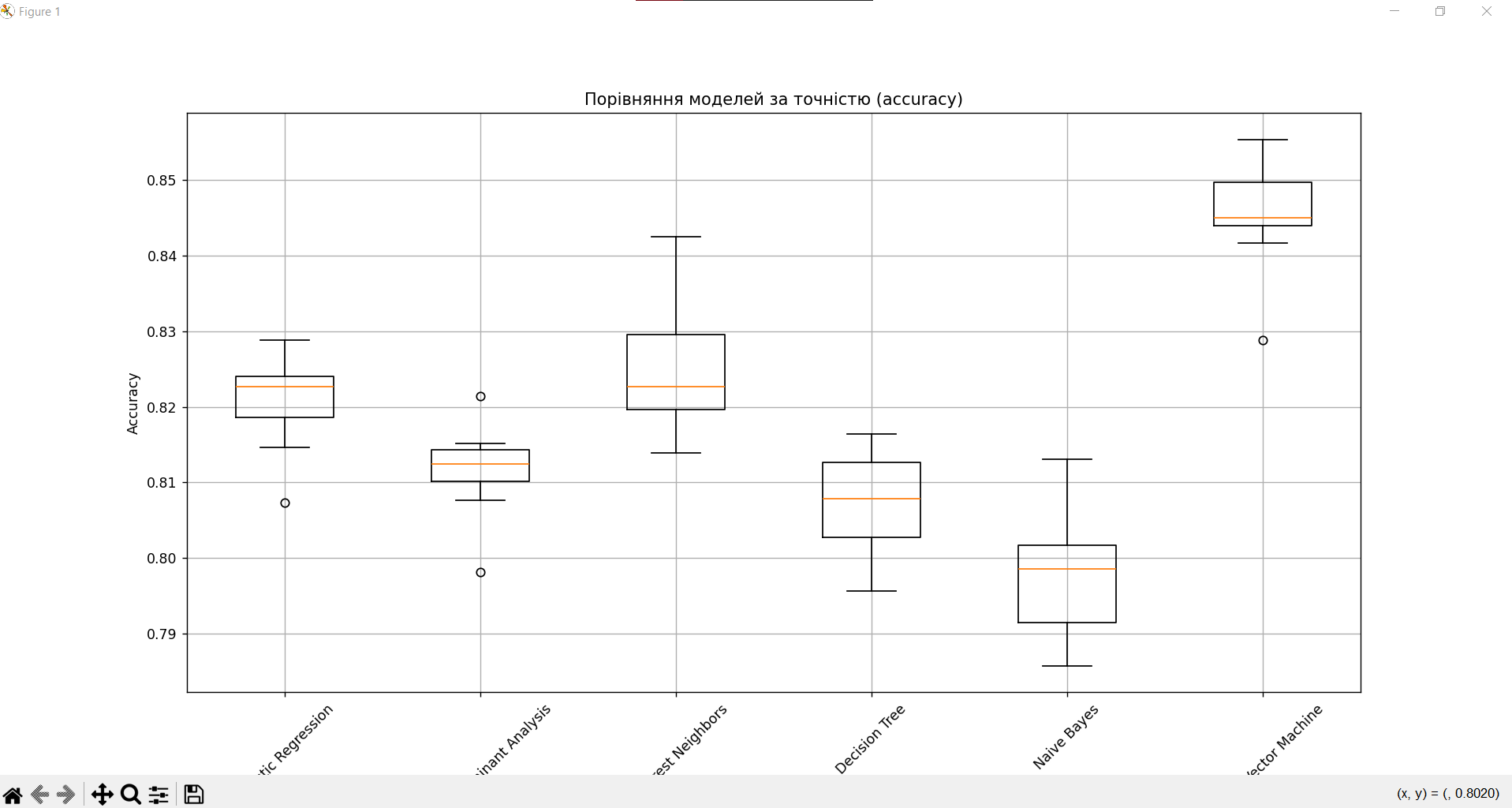


Рис. 12

Результат виконання програми:

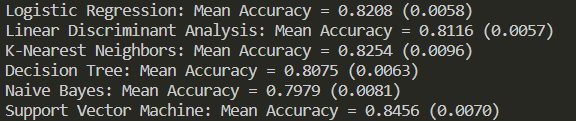


Рис. 13

В цій ситуації найкращий результат показав SVM, можна зробити висновок що для цієї задачі він підходить найкраще.



Лістинг:

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.linear\_model import RidgeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from io import BytesIO

from sklearn.metrics import confusion\_matrix

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(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('\nClassification Report:\n', metrics.classification\_report(y\_test, y\_pred))

mat = confusion\_matrix(y\_test, y\_pred)

sns.set()

sns.heatmap(mat.T, *square*=True, *annot*=True, *fmt*='d', *cbar*=False)

plt.xlabel('True Label')

plt.ylabel('Predicted Label')

plt.title("Confusion Matrix of Ridge Classifier")

plt.savefig("Confusion.jpg")

f = BytesIO()

plt.savefig(f, *format*="svg")

plt.show()

Результат виконання програми:

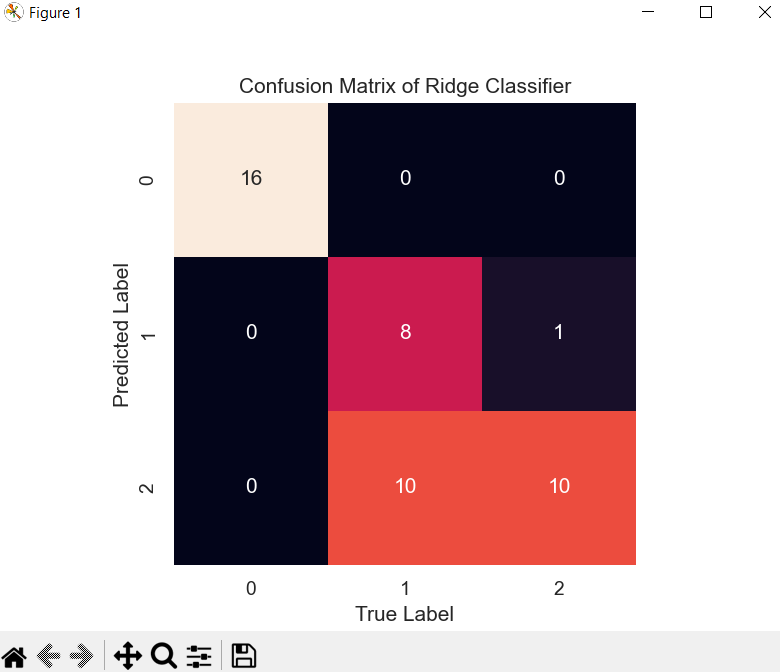


Рис. 14

Результат виконання програми:

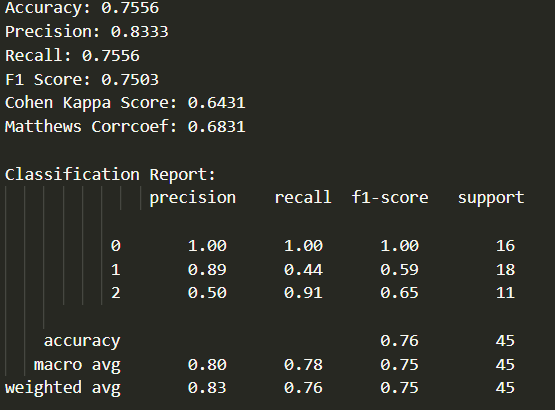


Рис. 15

Налаштування класифікатора Ridge:

1. tol=1e-2: Параметр tol визначає допустиму похибку для зупинки ітерацій алгоритму.

2. solver='sag': solver визначає метод розв’язання задачі оптимізації. 'sag' -Stochastic Average Gradient: є ітеративним методом, що ефективно працює на великих наборах даних. Він використовує стохастичний середній градієнт для мінімізації функції втрат.

Матриця плутанини показує наступне:

0: Класифікатор правильно передбачив всі 16 зразків.

1: З 18 зразків 8 було передбачено правильно, 10 зразків передбачено як клас 2, а 1 як клас 1.

2: Класифікатор правильно передбачив 10 зразків, 1 зразок було передбачено як клас 1.

Ця матриця плутанини показує, що класифікатор добре працює для класу 0, при цьому досить погано з іншими.

Коефіцієнт кореляції Метьюза – це оцінення якості матриці плутанини, отримане значення каже про те що модель демонструє не дуже хороший результат класифікування.

Коефіцієнта Каппа Коена – це ступінь збігів передбачень класифікатора з реальними мітками зазначених класів (враховуючи факт, що частина збігів це випадковість). Саме в цьому коді ми звіряємо показники передбачень з тестового набору ірисів.

Git: <https://github.com/PavlenkoOks/AI>