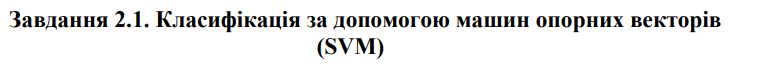
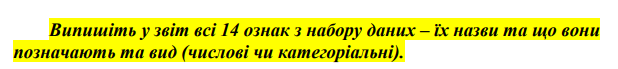
**ЛАБОРАТОРНА РОБОТА № 2**

**ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ**

**Хід роботи**

****

Ознайомтесь з набором даних.

****

39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical, Not-in-family, White, Male, 2174, 0, 40, United-States, <=50K

(В наборі насправді 15 ознак)

**Age (Вік)** – Тип: Числова

**Workclass (Клас роботи)** – Тип: Категоріальна

**fnlwgt (Фінальна вага)** – Тип: Числова

**Education** **(Освіта)** – Тип: Категоріальна

**Education-num** **(Кількість років освіти)** – Тип: Числова

**Marital-status** **(Сімейний стан)** – Тип: Категоріальна

**Occupation** **(Професія)** – Тип: Категоріальна

**Relationship** **(Тип родинних зв'язків)** – Тип: Категоріальна

**Race** **(Раса)** – Тип: Категоріальна

**Sex** **(Стать)** – Тип: Категоріальна

**Capital-gain** **(Капітальний дохід)** – Тип: Числова

**Capital-loss** **(Капітальні втрати)** – Тип: Числова

**Hours-per-week** **(Кількість годин на тиждень)** - Тип: Числова

**Native-country (Країна походження)** – Тип: Категоріальна

**Income (Дохід)** – 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[:-1].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)

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

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

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 (v2): {f1 \* 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:

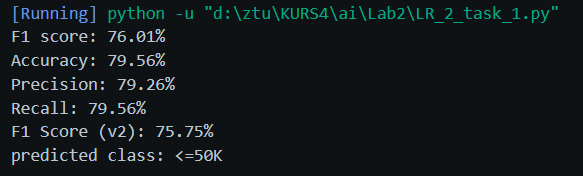
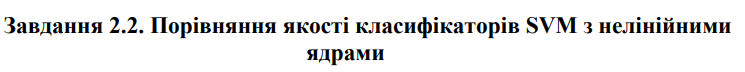


Рис.1

Зробіть висновок до якого класу належить тестова точка:

Тестова точка належить до класу: **Income (Дохід) - <=50K** (річний дохід менше або дорівнює $50,000).



Для зручнішої роботи з даними та загалом з кодом було написано окремий файл utilities.py, в якому зберігаються фрагменти коду які повторюються та можуть бути винесені в окрему функцію.

Лістинг:

import numpy as np

from sklearn import preprocessing

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

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

def read\_and\_prepare\_data(input\_file, max\_datapoints=25000):

    X = []

    Y = []

    count\_class1 = 0

    count\_class2 = 0

    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)

                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)

    print(f"Data successfully read.\nShape of array X: {X.shape}")

    return X, np.array(Y)

def encode\_categorical\_data(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)

    print(f"Encoding completed.\nShape of X after encoding: {X\_encoded.shape}")

    return X\_encoded, label\_encoder

def split\_data(X, Y, test\_size=0.2, random\_state=5):

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

    print("Data successfully split into training and testing sets.")

    return X\_train, X\_test, y\_train, y\_test

def calculate\_and\_print\_metrics(y\_test, y\_test\_pred):

    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}%")

**LR\_2\_task\_2\_1:**

import numpy as np

from sklearn.svm import SVC

from utilities import read\_and\_prepare\_data, encode\_categorical\_data, split\_data, calculate\_and\_print\_metrics

input\_file = 'income\_data.txt'

X, Y = read\_and\_prepare\_data(input\_file)

X\_encoded, label\_encoder = encode\_categorical\_data(X)

X = X\_encoded[:, :-1].astype(int)

Y = np.array(Y)

X\_train, X\_test, y\_train, y\_test = split\_data(X, Y)

print("Training model with polynomial kernel ('poly')")

classifier = SVC(kernel='poly', degree=2)

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

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

calculate\_and\_print\_metrics(y\_test, y\_test\_pred)

**LR\_2\_task\_2\_2:**

import numpy as np

from sklearn.svm import SVC

from utilities import read\_and\_prepare\_data, encode\_categorical\_data, split\_data, calculate\_and\_print\_metrics

input\_file = 'income\_data.txt'

X, Y = read\_and\_prepare\_data(input\_file)

X\_encoded, label\_encoder = encode\_categorical\_data(X)

X = X\_encoded[:, :-1].astype(int)

Y = np.array(Y)

X\_train, X\_test, y\_train, y\_test = split\_data(X, Y)

print("Training model with Gaussian kernel ('rbf')")

classifier = SVC(kernel='rbf')

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

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

calculate\_and\_print\_metrics(y\_test, y\_test\_pred)

**LR\_2\_task\_2\_3:**

import numpy as np

from sklearn.svm import SVC

from utilities import read\_and\_prepare\_data, encode\_categorical\_data, split\_data, calculate\_and\_print\_metrics

input\_file = 'income\_data.txt'

X, Y = read\_and\_prepare\_data(input\_file)

X\_encoded, label\_encoder = encode\_categorical\_data(X)

X = X\_encoded[:, :-1].astype(int)

Y = np.array(Y)

X\_train, X\_test, y\_train, y\_test = split\_data(X, Y)

print("Training model with sigmoid kernel ('sigmoid')")

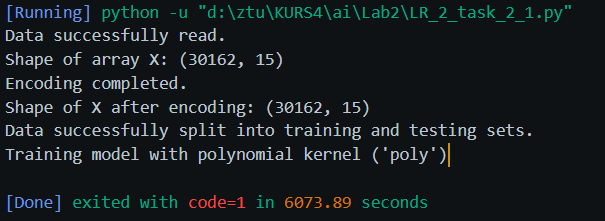
classifier = SVC(kernel='sigmoid')

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

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

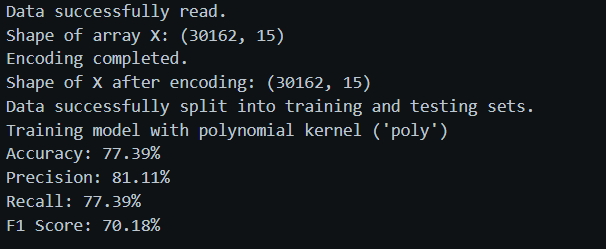
calculate\_and\_print\_metrics(y\_test, y\_test\_pred)

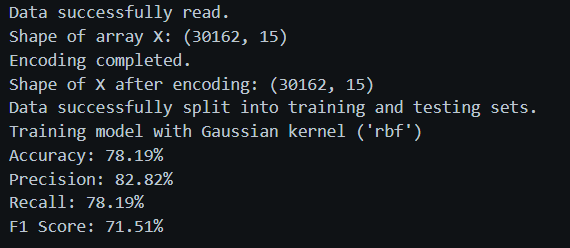
В ході виконання програми було виявлено що якщо використовувати degree=8, то виконання програми буде захмарно довгим:

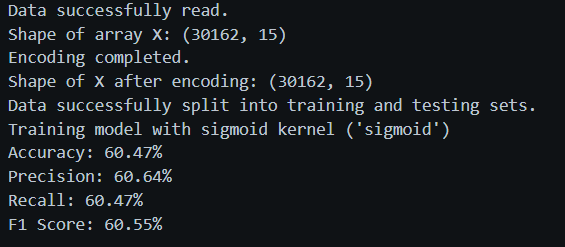


Оптимальним значенням було обрано 2

Результат виконання:

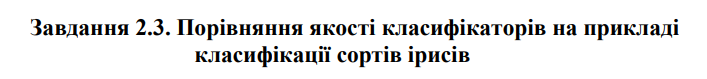






Найкращі результати для класифікації отримані з використанням **гаусового ядра (RBF)**, оскільки воно забезпечило найвищі показники точності, точності класифікації та F1-метрики. Тому, для цієї задачі класифікації рекомендовано використовувати гаусове ядро SVM.

Хоча можливо що при значенні degree=8 результати б були інші.





Лістинг:

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']))

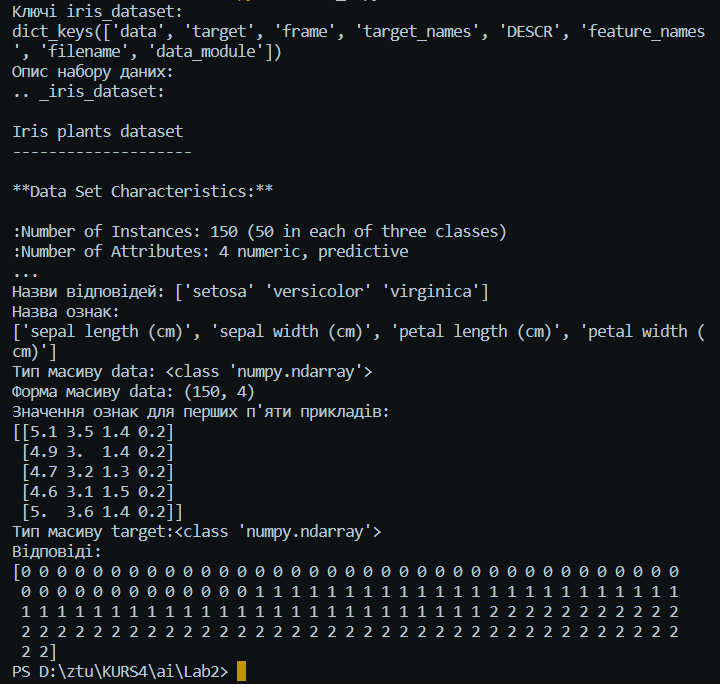
print("Тип масиву data: {}".format(type(iris\_dataset['data'])))

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']))





Лістинг:

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)

print(dataset.shape)

print(dataset.head(20))

print(dataset.describe())

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

dataset.plot(kind='box', subplots=True, layout=(2,2),

sharex=False, sharey=False)

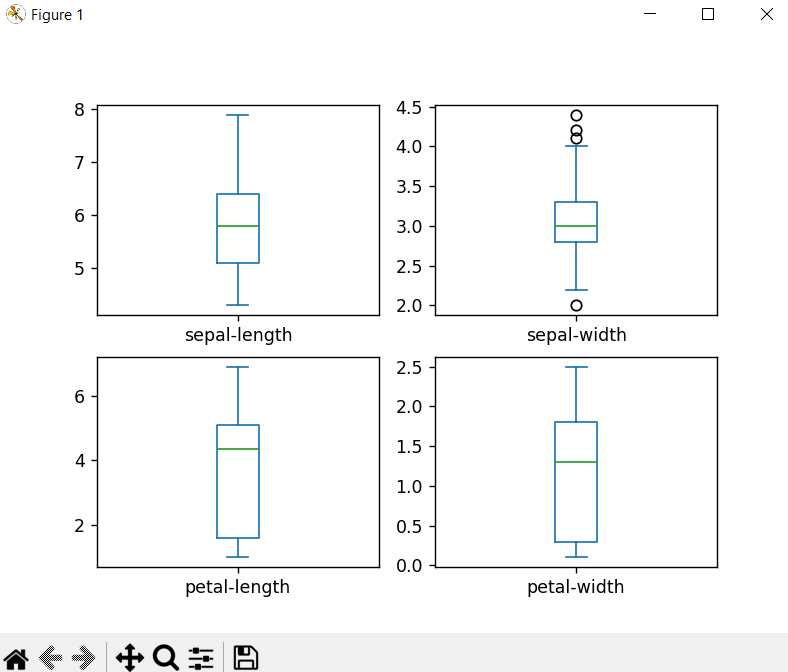
pyplot.show()

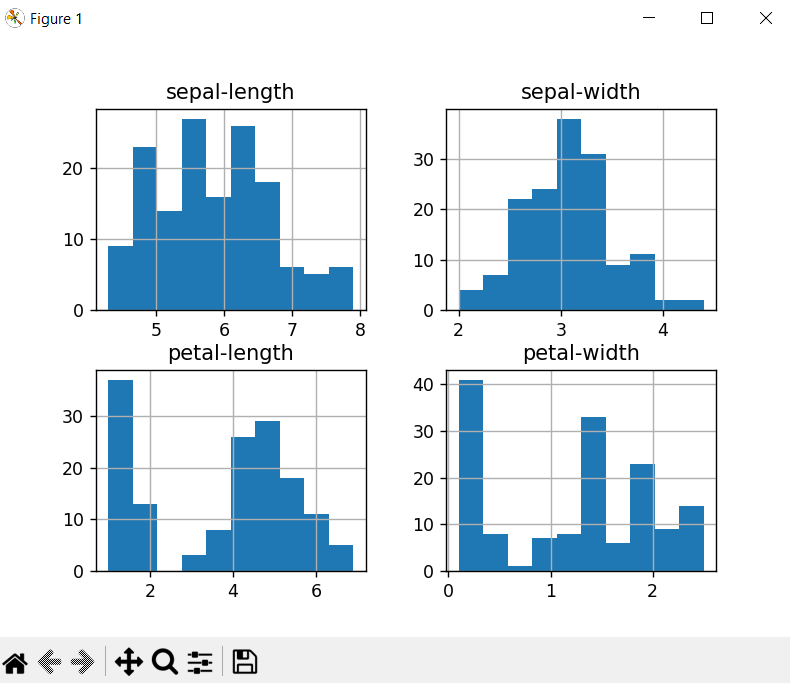
dataset.hist()

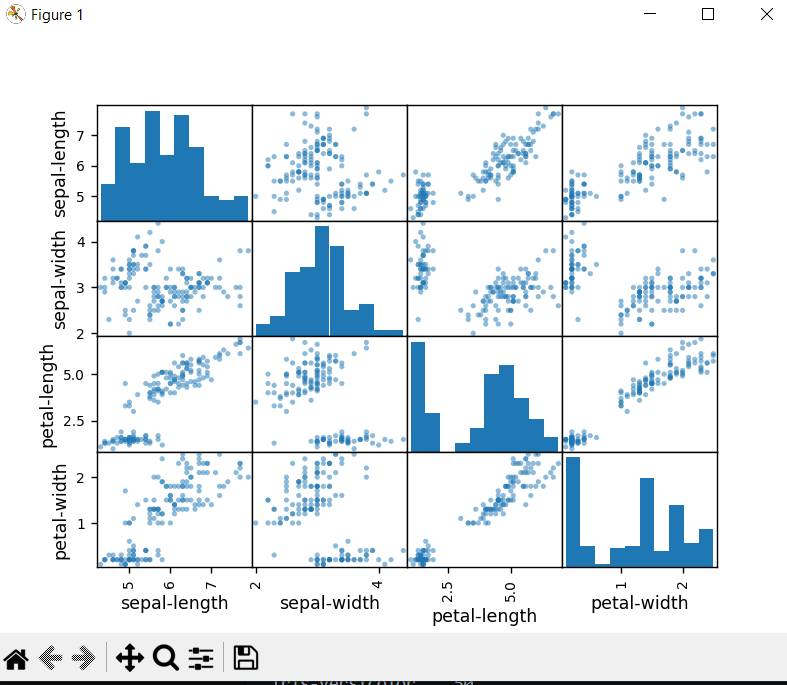
pyplot.show()

scatter\_matrix(dataset)

pyplot.show()

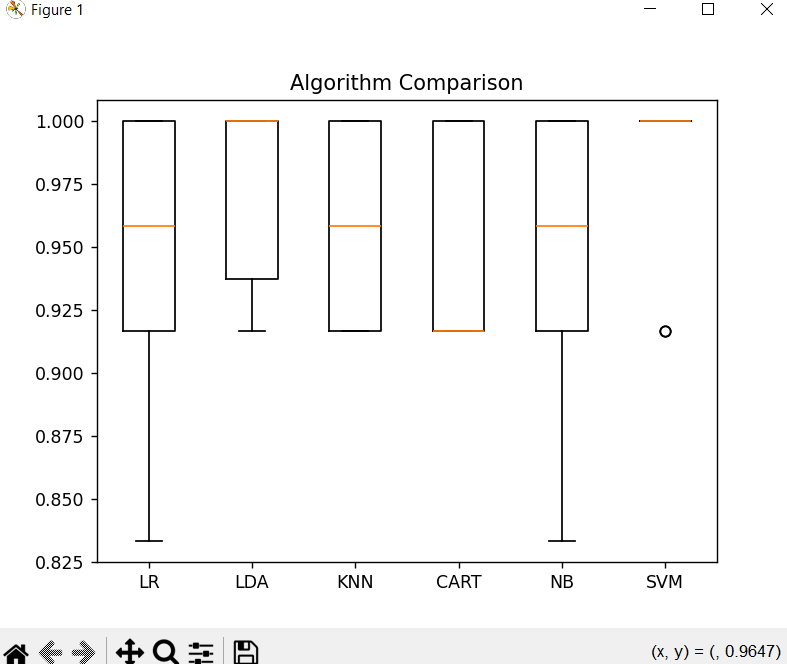


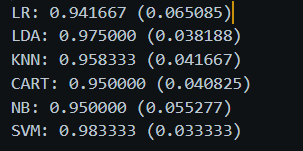


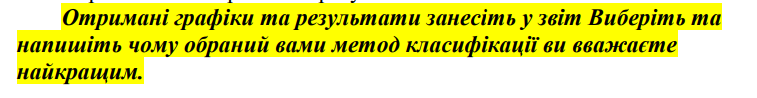










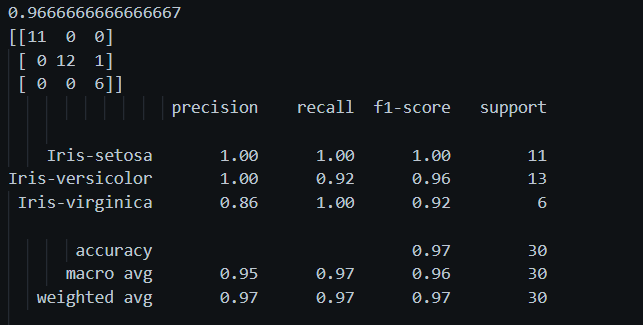


Мій вибір це SVM – він має найбільшу точність та найменше відхилення серед всіх.









Фінальний лістинг:

from pandas import read\_csv

from pandas.plotting import scatter\_matrix

from matplotlib import pyplot

import numpy as np

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\_info():

    print(dataset.shape)

    print(dataset.head(20))

    print(dataset.describe())

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

def print\_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()

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)

*# 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, tick\_labels=names)*

*# pyplot.title('Algorithm Comparison')*

*# pyplot.show()*

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]])

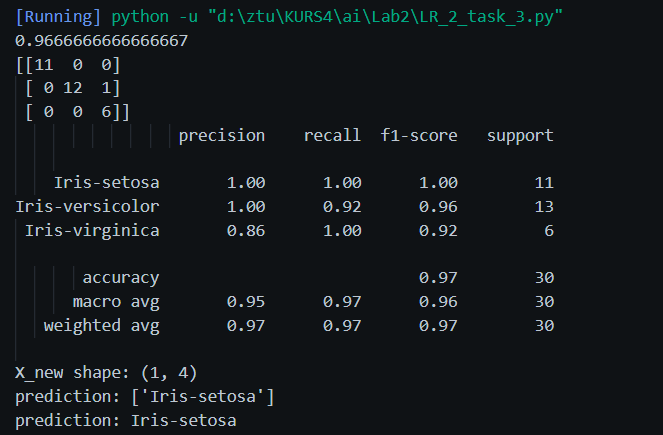
print("X\_new shape: {}".format(X\_new.shape))

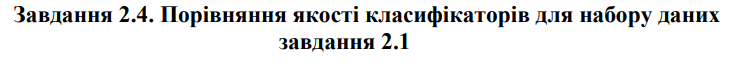
prediction = model.predict(X\_new)

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

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

Вид квітки та решта інформації вказана на рисунку:





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

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

from matplotlib import pyplot

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedKFold

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[:-1].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)

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

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:

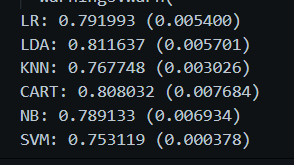
    kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)

    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()))



Можна зробити висновок що конкретно для цієї задачі краще підходить LinearDiscriminantAnalysis, він показав найвищий середній F1 Score серед усіх моделей. Це свідчить про його здатність добре класифікувати дані, враховуючи баланс між точністю та повнотою, а також низький рівень стандартного відхилення.



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")

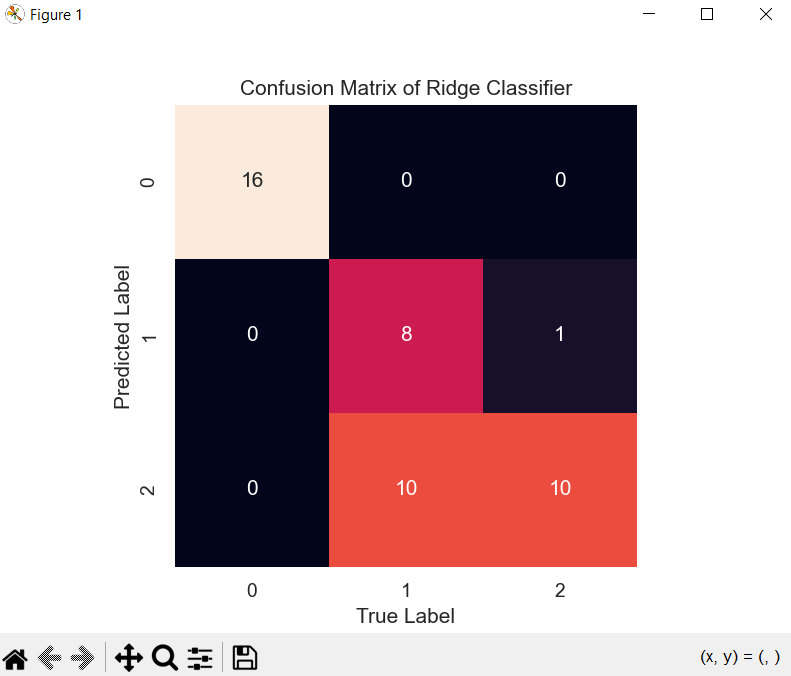
f = BytesIO()

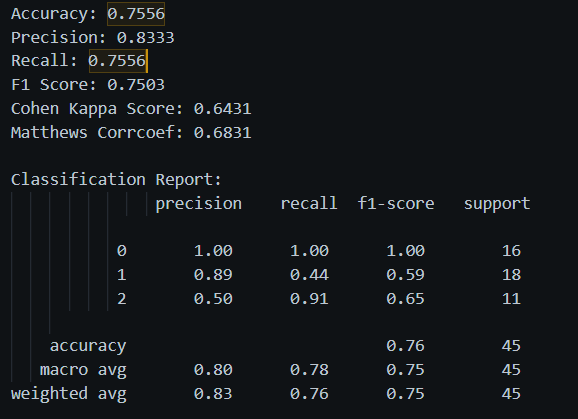
plt.savefig(f, format="svg")

plt.show()

Налаштування класифікатора Ridge:

1. **tol=1e-2**: Параметр tol визначає допустиму похибку для зупинки ітерацій алгоритму. Значення 1e-2 означає, що модель буде зупинятись, якщо зміни в похибці будуть менші за 0.01.
2. **solver='sag'**: solver визначає метод розв’язання задачі оптимізації. Вибір 'sag' (Stochastic Average Gradient) є ітеративним методом, що ефективно працює на великих наборах даних. Він використовує стохастичний середній градієнт для мінімізації функції втрат.





На матриці плутанини видно наступне:

* Класи:
  + 0: Класифікатор правильно передбачив всі 16 зразків.
  + 1: З 18 зразків 8 було передбачено правильно, 1 зразок було передбачено як клас 0, а 10 зразків передбачено як клас 2.
  + 2: Класифікатор правильно передбачив 10 зразків, 1 зразок було передбачено як клас 1.

Ця матриця плутанини показує, що класифікатор добре працює для класу 0, але має труднощі з відділенням класів 1 та 2.

Коефіцієнт кореляції Метьюза:

Це якість оцінення якості матриці плутанини, Отримане значення 0.6831 показує, що модель є достатньо якісною і демонструє добрий результат класифікування, але при цьому, як вже було зазначено вище, щось їй дається важче, тому результат хоч і не поганий, але точно не хороший.

Коефіцієнта Каппа Коена:

Розраховує ступінь збігів між передбаченнями класифікатора та реальними мітками класів, враховуючи той факт, що частина збігів може бути випадковою. У цьому випадку коефіцієнт Каппа Коена порівнює передбачення моделі Ridge Classifier із істинними класами тестового набору даних Iris. У відповіді досить високий рівень збігів, хоч і не ідеальний.

Git: <https://github.com/Alhim616/AI_Labs_Yanushevych>