ЛАБОРАТОРНА РОБОТА № 2

ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ

Лукашевич Влада ІПЗ-21-1

https://github.com/vladalukashevych/artificial-intelligence-systems

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

age	Feature	Integer	Age
workclass	Feature	Categorical	Income
education	Feature	Categorical	Education Level
education-num	Feature	Integer	Education Level
marital-status	Feature	Categorical	Other
occupation	Feature	Categorical	Other
relationship	Feature	Categorical	Other
race	Feature	Categorical	Race
sex	Feature	Binary	Sex
capital-gain	Feature	Integer	
capital-loss	Feature	Integer	
hours-per-week	Feature	Integer	
native-country	Feature	Categorical	Other
income	Target	Binary	Income

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

# 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[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)

# Створення SVM-класифікатора
classifier = OneVsOneClassifier(LinearSVC(random_state=0))
# Навчання класифікатора
classifier.fit(X, y)

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))
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([item])[0])
  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])



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 score: 76.12%
<=50K
Accuracy: 79.92%
Precision: 79.45%
Recall: 79.92%
```

За вихідними даними можемо зробити висновок, що тестова точка належить до класу <=50K.

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами

```python
import numpy as np
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score

# Читання даних
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 5000  # Спробуємо зменшити кількість даних для тестування

with open('income_data.txt', '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

# 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[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(float)  # Тепер використовуємо float, щоб було
легше нормалізувати
y = X_encoded[:, -1].astype(int)

# Нормалізація даних
scaler = preprocessing.StandardScaler()
X = scaler.fit_transform(X)

# Поділ даних на тренувальні та тестові вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)



classifier = SVC(kernel='poly', degree=3)  # degree=3 для поліноміального ядра
classifier.fit(X_train, y_train)

# Оцінка класифікатора за допомогою перехресної перевірки (cross-validation)
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy', cv=3)
precision_values = cross_val_score(classifier, X, y,
scoring='precision_weighted', cv=3)
recall_values = cross_val_score(classifier, X, y, scoring='recall_weighted',
cv=3)

print(f"Results for SVM with poly kernel:")
print("F1 score: " + str(round(100 * f1.mean(), 2)) + "%")
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
print("Recall: " + str(round(100 * recall_values.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([item])[0])
        count += 1

input_data_encoded = np.array(input_data_encoded)
input_data_encoded = scaler.transform(input_data_encoded.reshape(1, -1))  #
Нормалізуємо дані тестової точки

# Використання класифікатора для передбачення класу
predicted_class = classifier.predict(input_data_encoded)
print("Predicted class:", label_encoder[-
1].inverse_transform(predicted_class)[0])
```

```
Results for SVM with poly kernel:
F1 score: 77.95%
Accuracy: 78.24%
Precision: 79.75%
Recall: 78.24%
Predicted class: >50K
```

```python
import numpy as np
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score

# Читання даних
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 5000  # Спробуємо зменшити кількість даних для тестування

with open('income_data.txt', '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

# 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[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(float)  # Тепер використовуємо float, щоб було легше нормалізувати
y = X_encoded[:, -1].astype(int)

# Нормалізація даних
scaler = preprocessing.StandardScaler()
X = scaler.fit_transform(X)

# Поділ даних на тренувальні та тестові вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)



classifier = SVC(kernel='rbf')
classifier.fit(X_train, y_train)

# Оцінка класифікатора за допомогою перехресної перевірки (cross-validation)
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy', cv=3)
precision_values = cross_val_score(classifier, X, y, scoring='precision_weighted', cv=3)
recall_values = cross_val_score(classifier, X, y, scoring='recall_weighted', cv=3)

print(f"Results for SVM with rbf kernel:")
print("F1 score: " + str(round(100 * f1.mean(), 2)) + "%")
```

```
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
print("Recall: " + str(round(100 * recall_values.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([item])[0])
        count += 1

input_data_encoded = np.array(input_data_encoded)
input_data_encoded = scaler.transform(input_data_encoded.reshape(1, -1))  #
Нормалізуємо дані тестової точки

# Використання класифікатора для передбачення класу
predicted_class = classifier.predict(input_data_encoded)
print("Predicted class:", label_encoder[-
1].inverse_transform(predicted_class)[0])
```

```
Results for SVM with rbf kernel:
F1 score: 80.61%
Accuracy: 80.68%
Precision: 81.12%
Recall: 80.68%
Predicted class: <=50K
```

```
import numpy as np
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score

# Читання даних
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 5000  # Спробуємо зменшити кількість даних для тестування

with open('income_data.txt', '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

# 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[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(float)  # Тепер використовуємо float, щоб було
легше нормалізувати
y = X_encoded[:, -1].astype(int)

# Нормалізація даних
scaler = preprocessing.StandardScaler()
X = scaler.fit_transform(X)

# Поділ даних на тренувальні та тестові вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)



classifier = SVC(kernel='sigmoid')
classifier.fit(X_train, y_train)

# Оцінка класифікатора за допомогою перехресної перевірки (cross-validation)
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy', cv=3)
precision_values = cross_val_score(classifier, X, y,
scoring='precision_weighted', cv=3)
recall_values = cross_val_score(classifier, X, y, scoring='recall_weighted',
cv=3)

print(f"Results for SVM with sigmoid kernel:")
print("F1 score: " + str(round(100 * f1.mean(), 2)) + "%")
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
print("Recall: " + str(round(100 * recall_values.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([item])[0])
        count += 1

input_data_encoded = np.array(input_data_encoded)
input_data_encoded = scaler.transform(input_data_encoded.reshape(1, -1))  #
Нормалізуємо дані тестової точки

# Використання класифікатора для передбачення класу
```

```
predicted_class = classifier.predict(input_data_encoded)
print("Predicted class:", label_encoder[-
1].inverse_transform(predicted_class)[0])
```

```
Results for SVM with sigmoid kernel:
F1 score: 67.34%
Accuracy: 67.34%
Precision: 67.34%
Recall: 67.34%
Predicted class: <=50K
```

З результатів отриманих під час виконання кодів з різними видами SMV, можемо зробити висновок, що нелінійний класифікатор з гаусовим ядром працює найкраще. Його якісні показники видають значення 80% та вище, які є найвищими серед всіх.

Завдання 2.3. Порівняння якості класифікаторів на прикладі класифікації сортів ірисів

ЗАВАНТАЖЕННЯ ТА ВИВЧЕННЯ ДАНИХ

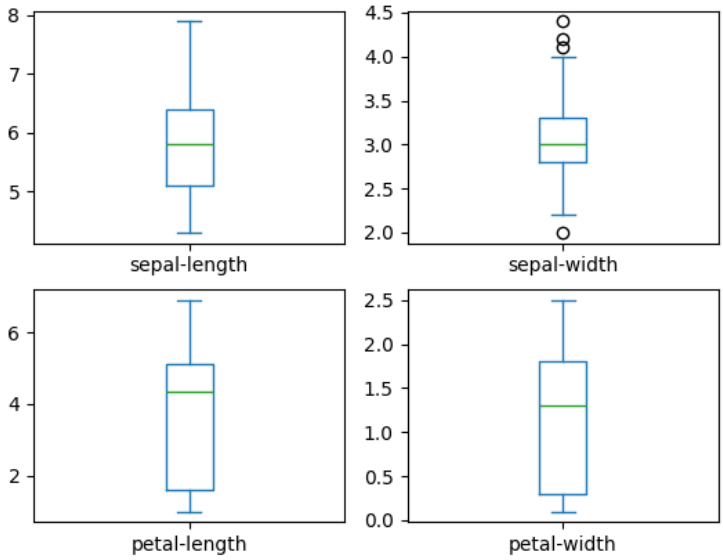
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()

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("Перші п'ять прикладів ознак:\n{}".format(iris_dataset['data'][:5]))

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

```
Ключі iris_dataset:
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
.. _iris_dataset:

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)
```

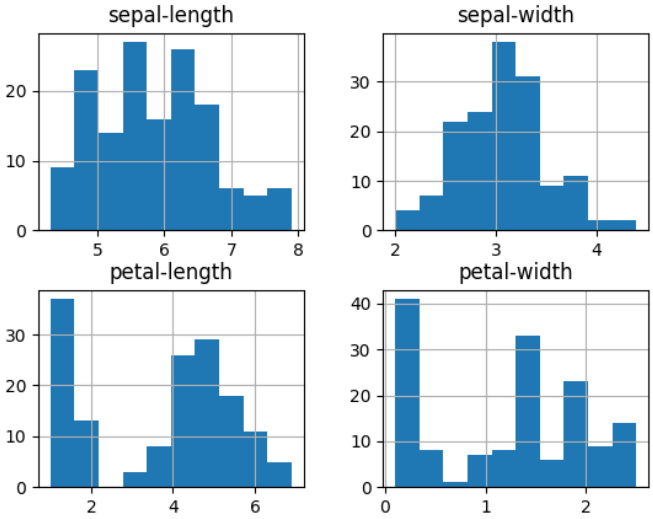
```
Перші п'ять прикладів ознак:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
Тип масиву target: <class 'numpy.ndarray'>
Відповіді:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 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
 2 2]
```

ВІЗУАЛІЗАЦІЯ ДАНИХ

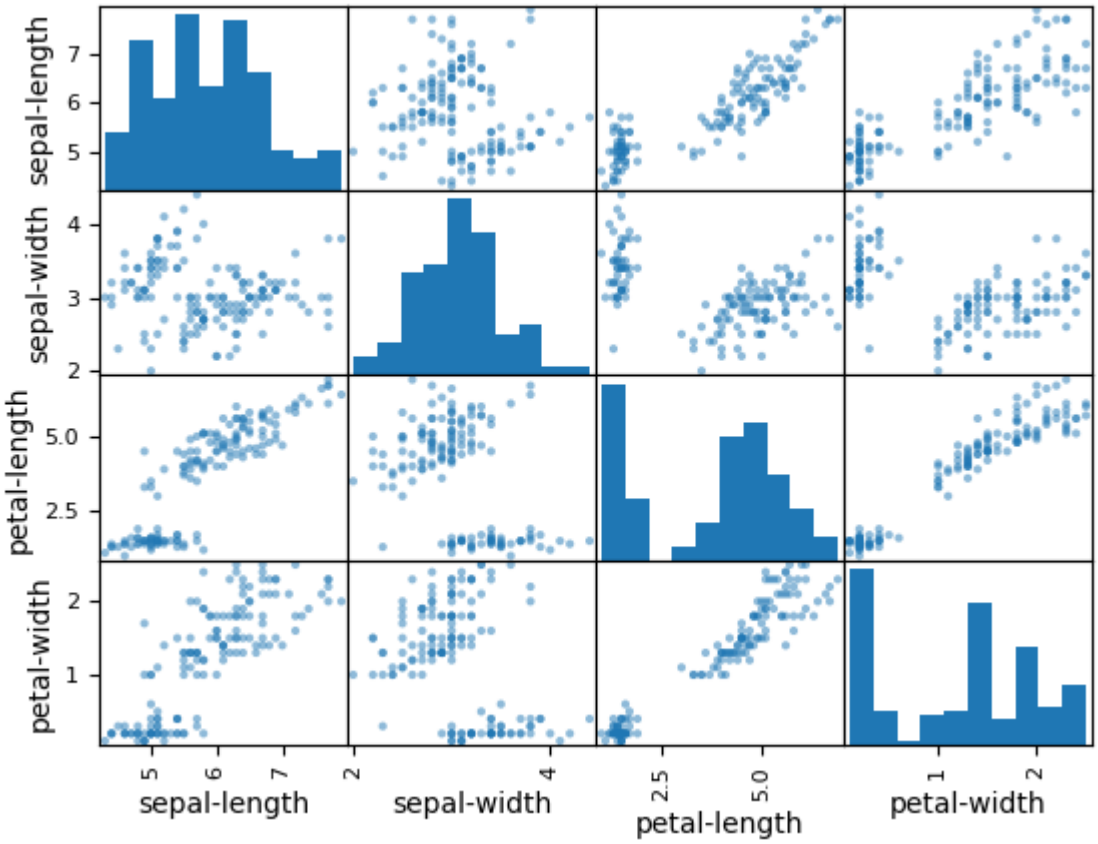
Діаграма розмаху



Гістограма розподілу атрибутів датасету



Матриця діаграм розсіювання

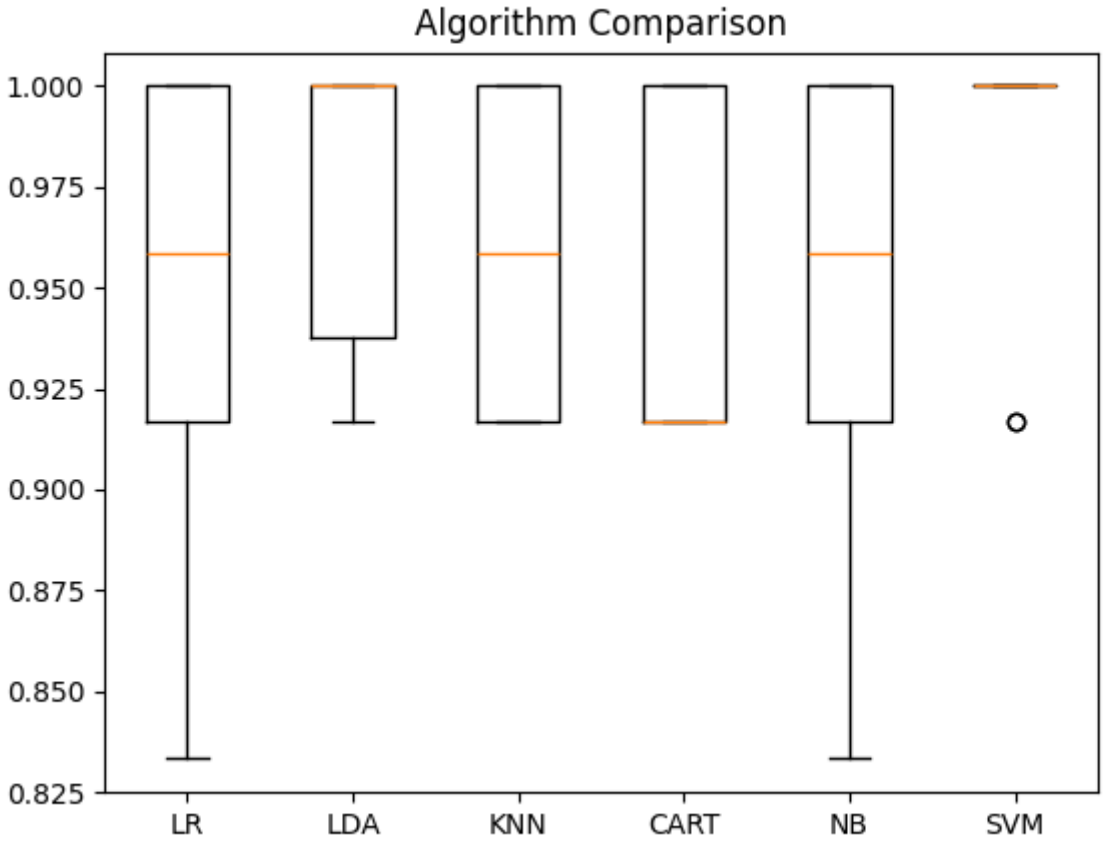


```
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
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)

# 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()
```

```
(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
```

КЛАСИФІКАЦІЯ (ПОБУДОВА МОДЕЛІ)




```
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)
```

SVM демонструє досить стабільну роботу, з майже відсутньою варіацією результатів, але присутній один винятковий випадок поза так званим боксом.

NB та LDA показують найбільші варіації у своїй продуктивності.

LR, KNN та CART мають відносно стабільну продуктивність з медіаною близько 0.95.

З нашим набором даних найкраще себе проявив алгоритм SVM, оскільки він продемонстрував найбільшу стабільність і точність результатів серед усіх розглянутих алгоритмів.

ОЦІНКА ЯКОСТІ МОДЕЛІ

```
Accuracy:  0.9666666666666667

Confusion Matrix:
 [[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]

Classification Report:
                 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
```

ОТРИМАННЯ ПРОГНОЗУ (ЗАСТОСУВАННЯ МОДЕЛІ ДЛЯ ПЕРЕДБАЧЕННЯ)

```
Форма масиву X_new: (1, 4)
Прогноз: Iris-setosa
```

```
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
import numpy as np


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)


# 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()


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

```
# Оцінюємо прогноз
print("Accuracy: ", accuracy_score(Y_validation, predictions))
print("Confusion Matrix:\n", confusion_matrix(Y_validation, predictions))
print("\nClassification Report:\n", classification_report(Y_validation,
predictions))



# Нові дані квітки
X_new = np.array([[5, 2.9, 1, 0.2]])
print("Форма масиву X_new: {}".format(X_new.shape))
prediction = model.predict(X_new)
print("Прогноз: {}".format(prediction[0]))
```

За результатами тренування найвищу точність класифікації у вигляді 98,3333% вдалося досягти використавши алгоритм SVM.

За результатами прогнозу квітка з кроку 8 належить до класу Iris-setosa.

Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1

```
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split, cross_val_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 sklearn.model_selection import StratifiedKFold
from sklearn.multiclass import OneVsRestClassifier

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

with open('income_data.txt', '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

# 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[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)


# Поділ даних на навчальну і тестову вибірку
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5)

# Завантажуємо алгоритми моделі
models = []
models.append(('LR',
OneVsRestClassifier(LogisticRegression(solver='liblinear'))))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))

# 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()))
```

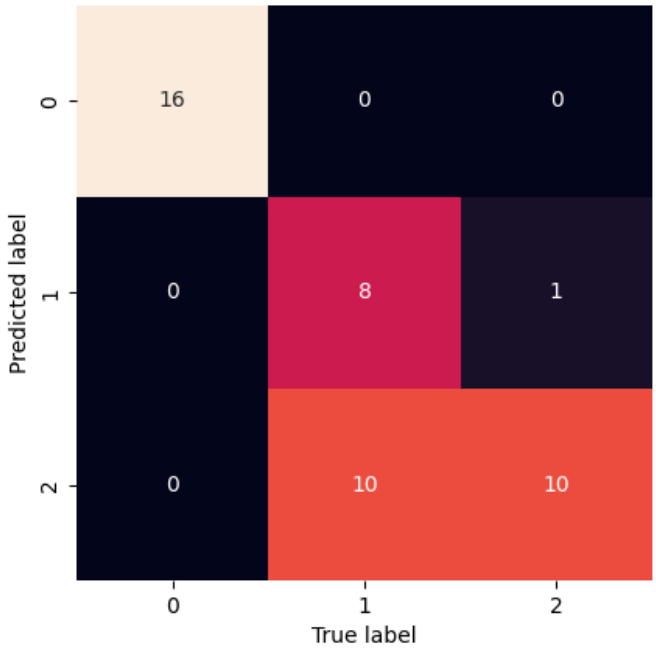
```
LR: 0.793651 (0.007915)
LDA: 0.811637 (0.005701)
KNN: 0.767748 (0.003026)
CART: 0.806913 (0.007850)
NB: 0.789133 (0.006934)
SVM: 0.788512 (0.002538)
```

Програмою було обраховано показники якості класифікації переліку алгоритмів. З отриманих даних ми можемо побачити, що найліпше з таким набором даних справився алгоритм LDA, так як його показник є найвищим – 81,1637%.

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

```
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

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)

classifier = RidgeClassifier(tol=1e-2, solver="sag")

classifier.fit(X_train, y_train)

y_pred = classifier.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 = metrics.confusion_matrix(y_test, y_pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('True label')
plt.ylabel('Predicted label')

plt.savefig("Confusion.jpg")

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

plt.show()
```

```
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
```



В цьому коді маємо наступні налаштування RidgeClassifier:

- tol=1e-2: параметр толерантності (tolerance) визначає точність вирішення оптимізаційної задачі. Менше значення цього параметра призводить до більш точного вирішення, але потребує більше обчислювальних ресурсів.

- solver="sag": цей параметр визначає алгоритм для оптимізації. У даному випадку використовується sag (Stochastic Average Gradient), який є оптимізатором на основі градієнта, ефективним для великих наборів даних.

Розраховані показники якості можна побачити вище.

Confusion Matrix (матриця неточностей, що є зображена вище) відображає порівняння фактичних і спрогнозованих класів. На вісі X знаходяться істинні мітки, на вісі Y – спрогнозовані. Чим більше число на діагоналі матриці, тим краще класифікатор справляється з передбаченням цього класу. У цій матриці ми бачимо, наскільки правильно модель класифікує кожен з трьох класів і наскільки часто вона допускає помилки.

Коефіцієнт Коена Каппа – це метрика, що використовується для вимірювання ступеня узгодженості між двома наборами класифікацій (фактичними та передбаченими) з поправкою на випадкову угоду. Вона важлива, коли класи присутні в даних не рівномірно.

Коефіцієнт Метьюза є мірою якості класифікації, яка враховує істинно позитивні, хибно позитивні, істинно негативні та хибно негативні передбачення. Він ефективний, коли класи в наборі даних незбалансовані.