Лабораторна робота №2

ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ

Мета: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити різні методи класифікації даних та навчитися їх порівнювати.

Github - https://github.com/Oleksi89/ai-systems

Хід роботи

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

age: вік (числовий)

workclass: клас роботодавця (категоріальний)

finlwgt: фінальна вага (кількість людей, яку представляє цей запис) (числовий)

education: рівень освіти (категоріальний)

education-num: кількість років навчання (числовий)

marital-status: сімейний стан (категоріальний)

occupation: професія/сфера діяльності (категоріальний)

relationship: родинні стосунки (категоріальний)

race: раса (категоріальний)

sex: стать (категоріальний)

capital-gain: приріст капіталу (числовий)

capital-loss: втрата капіталу (числовий)

hours-per-week: години роботи на тиждень (числовий)

native-country: країна походження (категоріальний)

цільова змінна: income (дохід <=50K або >50K)

Лістинг програми:

```
import numpy as np
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

# 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)

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)

# Навчання на тренувальних даних та прогноз
classifier = OneVsOneClassifier(LinearSVC(random_state=0))
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)
```

```
# Обчислення F-міри для SVM
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("F1 score: " + str(round(100 * f1.mean(), 2)) + "%")

# Обчислення показників якості класифікації на тестовому наборі
print("\n--- Metrics ---")
acc = accuracy_score(y_test, y_test_pred)
prec = precision_score(y_test, y_test_pred, average='weighted')
rec = recall_score(y_test, y_test_pred, average='weighted')
f1 = f1_score(y_test, y_test_pred, average='weighted')

print("Accuracy: " + str(round(100 * acc, 2)) + "%")
print("Precision: " + str(round(100 * prec, 2)) + "%")
print("Recall: " + str(round(100 * rec, 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:
        # Використовуємо відповідний енкодер, reshape потрібен для transform
        input_data_encoded[i] = int(label_encoder[count].transform([input_data[i]])[0])
        count += 1

input_data_encoded = np.array(input_data_encoded)

# Використання класифікатора для кодованої точки даних та виведення результату
# Reshape data для передбачення одного зразка
predicted_class = classifier.predict(input_data_encoded.reshape(1, -1))

# Декодування результату
result_label = label_encoder[-1].inverse_transform(predicted_class)[0]
print("\nPredicted class: " + result_label)
```

Результати:

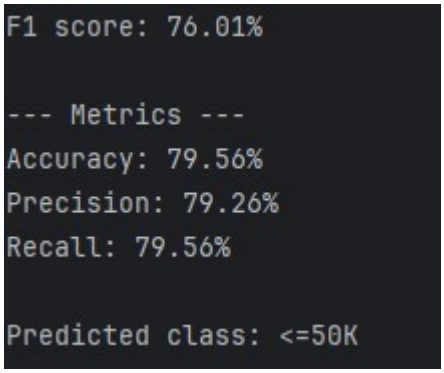

Рис.1

Показники якості класифікації:

F1-міра (F1 Score): 76.01%

Акуратність (Accuracy): 79.56%

Точність (Precision): 79.26%

Повнота (Recall): 79.56%

Висновок: На основі проведеної класифікації тестова точка даних належить до класу <=50K (дохід менше або дорівнює 50,000$).

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами з поліноміальним ядром

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing
from sklearn.multiclass import OneVsOneClassifier
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

# 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)

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)


# Створення SVM-класифікатора з поліноміальним ядром, навчання та прогноз
classifier = OneVsOneClassifier(SVC(kernel='poly', degree=8, max_iter=5000))
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# Обчислення F-міри для SVM
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("Poly kernel...")
print("F1 score: " + str(round(100 * f1.mean(), 2)) + "%")

# Обчислення показників якості класифікації на тестовому наборі
print("\n--- Metrics ---")
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average='weighted')
rec = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy: " + str(round(100 * acc, 2)) + "%")
print("Precision: " + str(round(100 * prec, 2)) + "%")
print("Recall: " + str(round(100 * rec, 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:
        # Використовуємо відповідний енкодер, reshape потрібен для transform
        input_data_encoded[i] = int(label_encoder[count].transform([input_data[i]])[0])
        count += 1

input_data_encoded = np.array(input_data_encoded)

# Використання класифікатора для кодованої точки даних та виведення результату
# Reshape data для передбачення одного зразка
predicted_class = classifier.predict(input_data_encoded.reshape(1, -1))

# Декодування результату
result_label = label_encoder[-1].inverse_transform(predicted_class)[0]
print("\nPredicted class: " + result_label)
```

Результати:

```
Poly kernel...
F1 score: 46.5%

--- Metrics ---
Accuracy: 25.59%
Precision: 80.98%
Recall: 25.59%

Predicted class: >50K
```

Рис.2

з гаусовим ядром

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing
from sklearn.multiclass import OneVsOneClassifier
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

# 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)

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)



# Створення SVM-класифікатора з гаусовим ядром, навчання та прогноз
classifier = OneVsOneClassifier(SVC(kernel='rbf', max_iter=5000))
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# Обчислення F-міри для SVM
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("Gausian kernel...")
print("F1 score: " + str(round(100 * f1.mean(), 2)) + "%")

# Обчислення показників якості класифікації на тестовому наборі
print("\n--- Metrics ---")
```

```
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average='weighted')
rec = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy: " + str(round(100 * acc, 2)) + "%")
print("Precision: " + str(round(100 * prec, 2)) + "%")
print("Recall: " + str(round(100 * rec, 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:
        # Використовуємо відповідний енкодер, reshape потрібен для transform
        input_data_encoded[i] = int(label_encoder[count].transform([input_data[i]])[0])
        count += 1

input_data_encoded = np.array(input_data_encoded)

# Використання класифікатора для кодованої точки даних та виведення результату
# Reshape data для передбачення одного зразка
predicted_class = classifier.predict(input_data_encoded.reshape(1, -1))

# Декодування результату
result_label = label_encoder[-1].inverse_transform(predicted_class)[0]
print("\nPredicted class: " + result_label)
```

Результати:

```
Gausian kernel...
F1 score: 71.95%

--- Metrics ---
Accuracy: 33.02%
Precision: 64.86%
Recall: 33.02%

Predicted class: >50K
```

Рис.2

з сигмоїдальним ядром

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing
from sklearn.multiclass import OneVsOneClassifier
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

# 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)

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)
```

```
print("Training sigmoid kernel...")
# Створення SVM-класифікатора з поліноміальним ядром, навчання та прогноз
classifier = OneVsOneClassifier(SVC(kernel='sigmoid', max_iter=5000))
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# Обчислення F-міри для SVM
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("Sigmoid kernel...")
print("F1 score: " + str(round(100 * f1.mean(), 2)) + "%")

# Обчислення показників якості класифікації на тестовому наборі
print("\n--- Metrics ---")
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average='weighted')
rec = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy: " + str(round(100 * acc, 2)) + "%")
print("Precision: " + str(round(100 * prec, 2)) + "%")
print("Recall: " + str(round(100 * rec, 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:
        # Використовуємо відповідний енкодер, reshape потрібен для transform
        input_data_encoded[i] = int(label_encoder[count].transform([input_data[i]])[0])
        count += 1

input_data_encoded = np.array(input_data_encoded)

# Використання класифікатора для кодованої точки даних та виведення результату
# Reshape data для передбачення одного зразка
predicted_class = classifier.predict(input_data_encoded.reshape(1, -1))

# Декодування результату
result_label = label_encoder[-1].inverse_transform(predicted_class)[0]
print("\nPredicted class: " + result_label)
```

Результати:

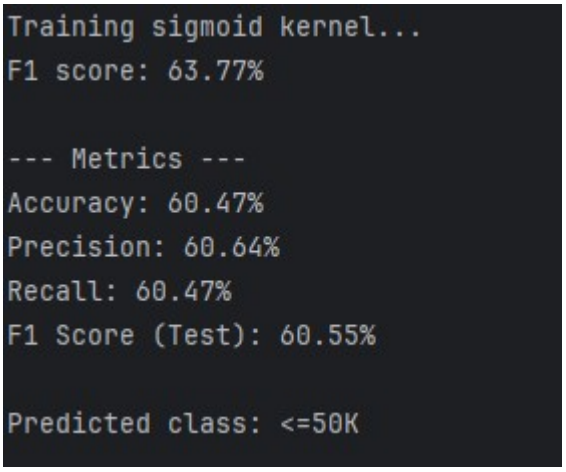

Рис.2

1. Поліноміальне ядро:

F1 Score: 46.5%

Accuracy (Акуратність): 25.59%

Precision (Точність): 80.98%

Recall (Повнота): 25.59%

Передбачений клас для тестової точки: >50K

2. Гаусове ядро:

F1 Score: 71.95%

Accuracy (Акуратність): 33.02%

Precision (Точність): 64.86%

Recall (Повнота): 33.02%

Передбачений клас для тестової точки: >50K

3. Сигмоїдальне ядро:

F1 Score: 63.77%

Accuracy (Акуратність): 60.47%

Precision (Точність): 60.64%

Recall (Повнота): 60.47%

Передбачений клас для тестової точки: <=50K

Висновок: За результатами тренування моделей з різними ядрами були отримані суперечливі дані. За показником F1-міри найкращий результат показало Гаусове ядро (71.95%). Однак, за показником Акуратності (Accuracy) найкраще впорався класифікатор із Сигмоїдальним ядром (60.47%), тоді як поліноміальне та

гаусове ядра показали низьку точність на тестовій вибірці (25% та 33% відповідно).

Тестова точка була класифікована по-різному:

Поліноміальне та Гаусове ядра віднесли її до класу >50K.

Сигмоїдальне ядро віднесло її до класу <=50K.

Завдання 2.3 Порівняння якості класифікаторів на прикладі класифікації сортів ірисів

Лістинг програми:

```
import matplotlib
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
import matplotlib
matplotlib.use('TkAgg')
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
from sklearn.multiclass import OneVsRestClassifier
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())

# крок2
# 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()

# крок3
# 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)

# крок4
# 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(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, tick_labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()

# крок6
# Створюємо прогноз на контрольній вибірці
model = SVC(gamma='auto')
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

# крок7
# Оцінюємо прогноз
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
```

```
print(classification_report(Y_validation, predictions))

# крок8
# нові дані: довжина, ширина, довжина пелюстки, ширина
X_new = np.array([[5, 2.9, 1, 0.2]])
print("форма массива X_new: {}".format(X_new.shape))

# Робимо прогноз
prediction = model.predict(X_new)
print("Прогноз: {}".format(prediction))
print("Спрогнозована мітка: {}".format(prediction[0]))
```

Результати:

```
(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
```

Рис.2

```
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
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      0.92      0.96        13
 Iris-virginica       0.86      1.00      0.92         6

       accuracy                           0.97        30
      macro avg       0.95      0.97      0.96        30
   weighted avg       0.97      0.97      0.97        30

форма массива X_new: (1, 4)
Прогноз: ['Iris-setosa']
Спрогнозована мітка: Iris-setosa
```

Рис.2

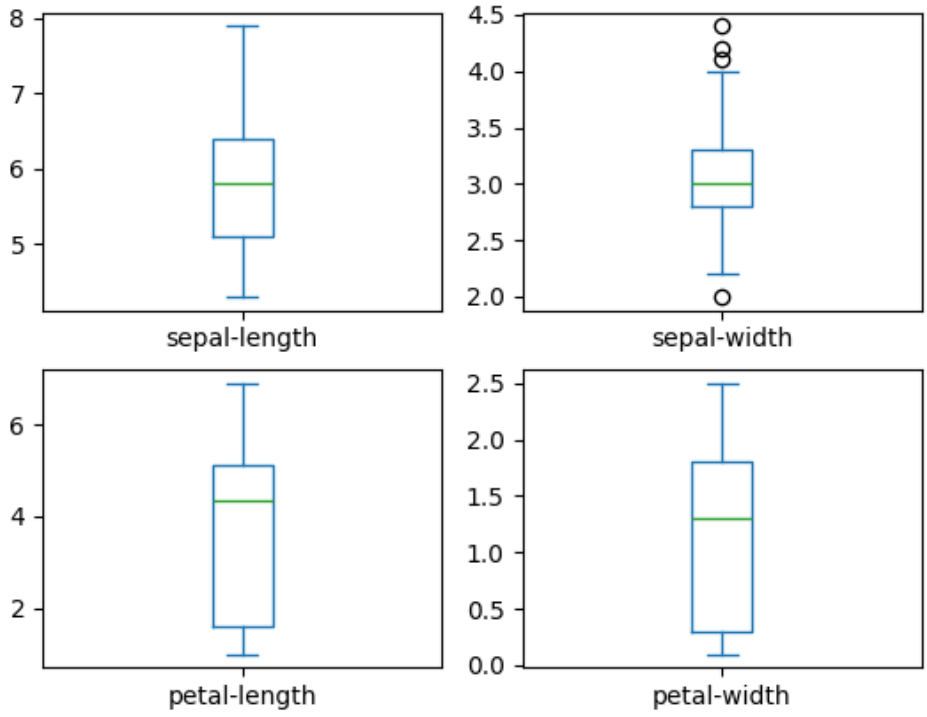

Рис.2 Діаграма розмаху атрибутів вхідних даних

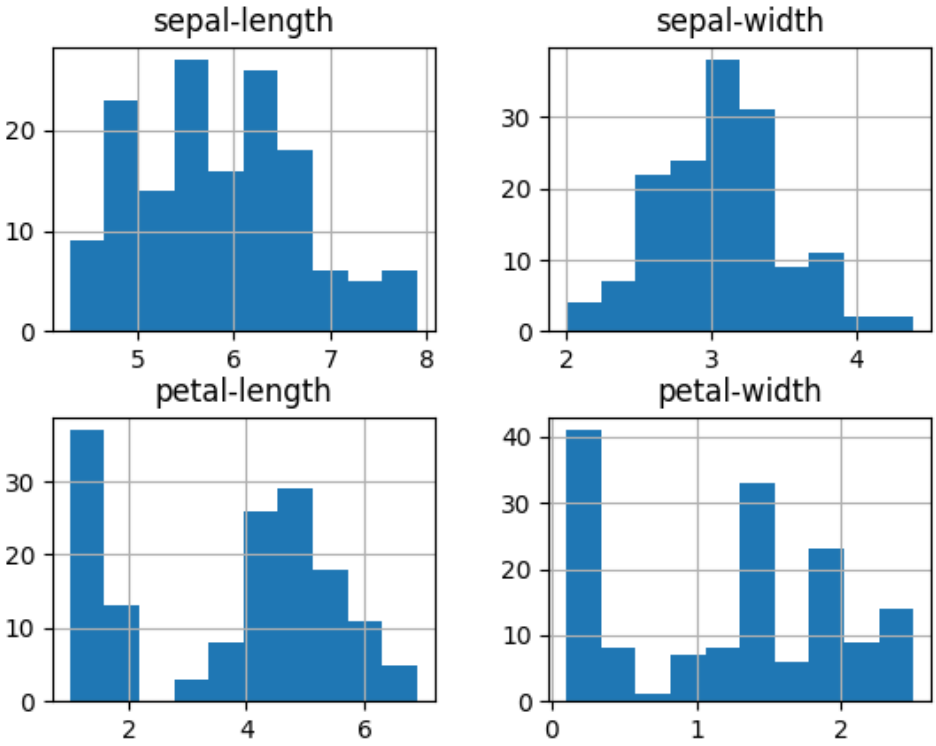

Рис.2 Гістограма розподілу атрибутів датасета

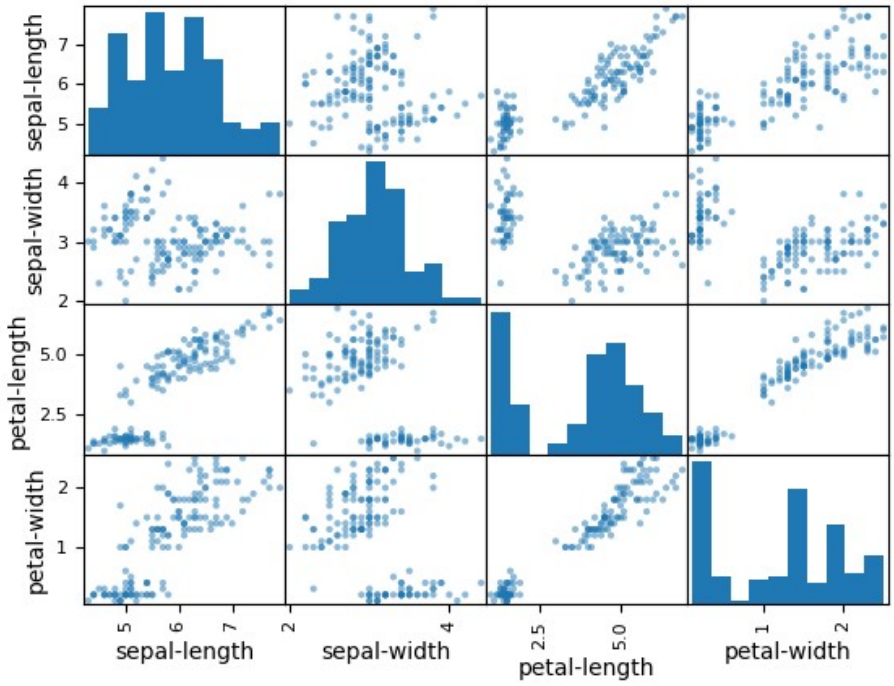

Рис.2 Матриця діаграм розсіювання

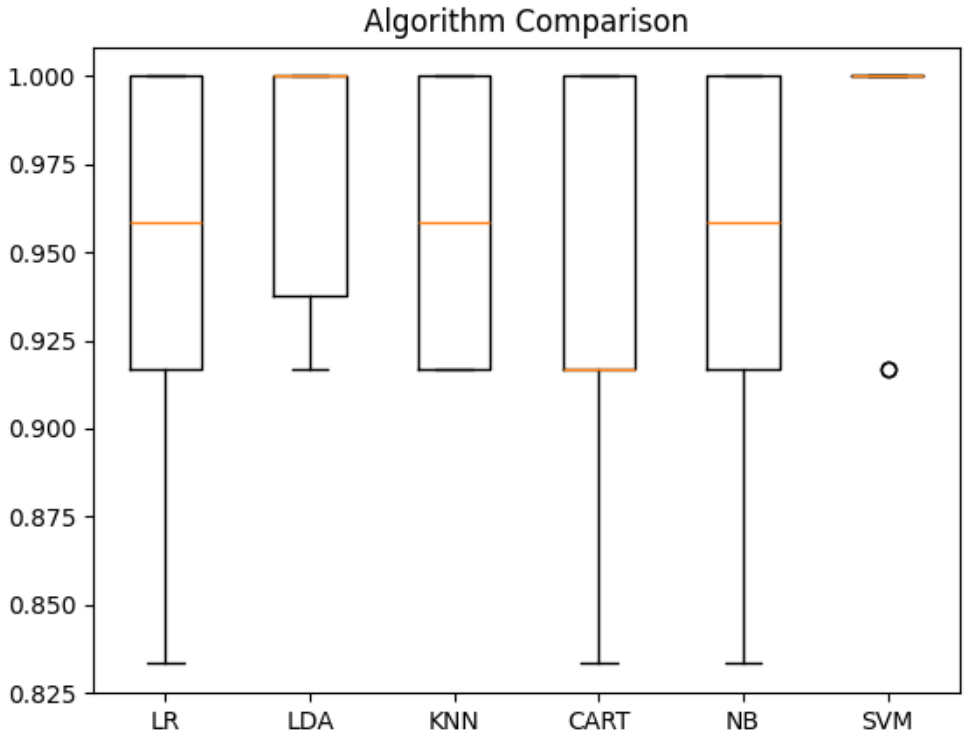

Рис.2 Порівняння алгоритмів

За результатами порівняння шести алгоритмів найкращу якість класифікації під час тренування (крос-валідації) показав метод SVM із середньою точністю 0.983333 (98.3%). На контрольній вибірці загальна точність склала 0.97 .

За результатами тренування та перевірки на контрольній вибірці вдалося досягти високої якості класифікації з точністю Accuracy = 0.97 (97%). Матриця плутанини показує, що з 30 тестових екземплярів модель допустила лише одну помилку (один зразок класу Iris-versicolor був помилково визначений як Iris-virginica), тоді як клас Iris-setosa визначається зі 100% точністю.

Квітка з кроку 8 (з параметрами: 5, 2.9, 1, 0.2) належить до класу Iris-setosa.

Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1 .

Лістинг програми:

```
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
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

# 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)

# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)

# Список алгоритмів для перевірки
models = []
models.append(('LR', LogisticRegression(solver='liblinear')))
```

```
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

# Змінні для пошуку найкращої моделі
best_model_name = ""
best_accuracy = 0.0

print("--- Порівняння результатів класифікації ---")

for name, model in models:
    # Навчання
    model.fit(X_train, y_train)

    # Прогноз
    y_pred = model.predict(X_test)

    # Розрахунок метрик
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted', zero_division=0)
    rec = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    # Вивід результатів
    print(f"{name}: Accuracy={acc:.2%}, Precision={prec:.2%}, Recall={rec:.2%}, F1={f1:.2%}")

    # Перевірка на найкращу точність
    if acc > best_accuracy:
        best_accuracy = acc
        best_model_name = name

print(f"\nНайкращий алгоритм: {best_model_name} з точністю {best_accuracy:.2%}")
```

Результати:

```
--- Порівняння результатів класифікації ---
LR: Accuracy=78.55%, Precision=76.88%, Recall=78.55%, F1=75.33%
LDA: Accuracy=81.12%, Precision=79.96%, Recall=81.12%, F1=79.49%
KNN: Accuracy=76.78%, Precision=74.31%, Recall=76.78%, F1=74.27%
CART: Accuracy=80.62%, Precision=80.84%, Recall=80.62%, F1=80.73%
NB: Accuracy=78.95%, Precision=77.43%, Recall=78.95%, F1=75.91%
SVM: Accuracy=74.41%, Precision=62.74%, Recall=74.41%, F1=63.59%

Найкращий алгоритм: LDA з точністю 81.12%
```

Рис.2

Було протестовано 6 алгоритмів класифікації. Отримані наступні показники:

LR: Accuracy=78.55%, F1=75.33%

LDA : Accuracy=81.12%, F1=79.49%

KNN: Accuracy=76.78%, F1=74.27%

CART: Accuracy=80.62%, F1=80.73%

NB: Accuracy=78.95%, F1=75.91%

SVM: Accuracy=74.41%, F1=63.59%

На основі отриманих метрик найкращим алгоритмом для вирішення задачі прогнозування меж доходу (бінарної класифікації) виявився Лінійний дискримінантний аналіз (LDA).

Я обрав цей алгоритм, тому що він показав найвищу точність класифікації 81.12% серед усіх протестованих моделей. Також він має високий показник F1-score 79.49%, що свідчить про хороший баланс між точністю (Precision) та повнотою Recall.

Другим за якістю став алгоритм дерев рішень CART з мінімальним відставанням 80.62%. Алгоритм SVM у даному випадку показав найгірший результат 74.41%, що робить його менш придатним для цього конкретного набору даних без додаткового налаштування параметрів.

Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

Лістинг програми:

```
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
from sklearn.metrics import confusion_matrix
from io import BytesIO
import seaborn as sns
import matplotlib.pyplot as plt

# Налаштування стилю
sns.set()

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

# Розділення даних
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.3, random_state=0)
```

```
# Створення та навчання класифікатора
# tol - допуск для зупинки, solver - метод розв'язання
clf = RidgeClassifier(tol=1e-2, solver="sag")
clf.fit(Xtrain, ytrain)

# Прогнозування
ypred = clf.predict(Xtest)

# Виведення метрик
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(ytest, ypred))

# Побудова матриці плутанини
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")

# Збереження SVG
f = BytesIO()
plt.savefig(f, format="svg")
plt.show()
```

Результати:

```
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
```

Рис.2

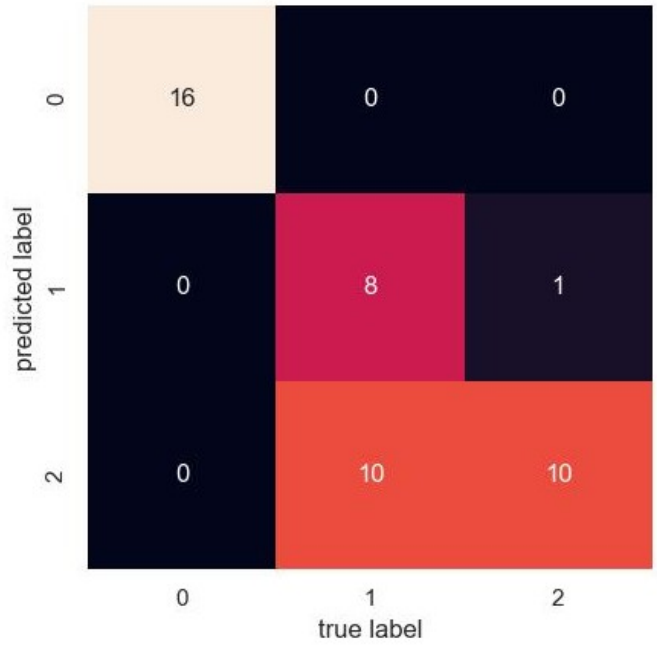

Рис.2

Налаштування Ridge: використано клас RidgeClassifier з наступними параметрами:

tol=1e-2 параметр допуску (0.01). Він визначає критерій зупинки алгоритму навчання: якщо на черговій ітерації покращення результату менше за це значення, навчання припиняється.

solver=”sag” алгоритм оптимізації (розв'язувач), який використовується для знаходження ваг моделі. Метод SAG є ефективним варіантом градієнтного спуску, що добре підходить для швидкого навчання.

Показники якості :

Отримані результати:

Accuracy (0.7556): Загальна точність моделі становить -76%. Це означає, що модель помиляється у чверті випадків.

Precision (0.8333): Зважена точність (здатність не позначати хибні зразки як позитивні) досить висока.

Recall (0.7556): Повнота (здатність знайти всі позитивні зразки) нижча, що свідчить про пропуски певних класів.

F1 Score (0.7503): Гармонічне середнє, що показує загальний баланс якості.

Пояснення зображення Confusion.jpg: Матриця плутанини показує, де саме помиляється алгоритм:

True label 0 (Setosa): Класифіковано ідеально. 16 зразків, всі 16 передбачені як клас 0.

True label 1 (Versicolor): Тут найбільша проблема. Зі всіх зразків класу 1 правильно визначено лише 8. Аж 10 зразків модель помилково віднесла до класу 2 (Predicted label 2).

True label 2 (Virginica): Правильно визначено 10 зразків. 1 зразок помилково віднесено до класу 1.

Висновок за матрицею: Лінійний класифікатор Ridge із цими налаштуваннями погано розділяє класи 1 та 2, сильно плутаючи їх (зсув у бік класу 2).

Коефіцієнт Коена Каппа та кореляція Метьюза:

Коефіцієнт Коена Каппа (0.6431) статистика, яка оцінює узгодженість передбачень з реальністю, виключаючи ймовірність випадкового вгадування. Значення 0.64 вказує на «суттєву» але не ідеальну узгодженість (далеко від 1).

Кореляція Метьюза (0.6831) більш надійна метрика для незбалансованих класів. Вона враховує всі чотири типи результатів (TP, TN, FP, FN). Значення 0.68 показує, що між прогнозом і фактом є позитивна кореляція вище середнього, але модель працює не ідеально через плутанину між сортами Versicolor та Virginica.

Висновок: у цій лабораторній роботі я використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив різні методи класифікації даних та навчитися їх порівнювати.