ЛАБОРАТОРНА РОБОТА № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

Мета: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити попередню обробку та класифікацію даних.

Посилання на GitHub: https://github.com/romankovban/ipz202_Kovban

Хід роботи:

Завдання 2.1. Попередня обробка даних

Як правило, при обробці ми маємо справу з великими обсягами необроблених вихідних даних. Алгоритми машинного навчання розраховані на те, що, перш ніж вони зможуть розпочати процес тренування, отримані дані будуть відформатовані певним чином. Щоб привести дані до форми, що прийнятна для алгоритмів машинного навчання, ми повинні попередньо підготувати їх і перетворити на потрібний формат.

2.1.1. Бінарізація

Цей процес застосовується в тих випадках, коли ми хочемо перетворити наші числові значення на булеві значення (0, 1). Скористаємося вбудованим методом для бінаризації вхідних даних, встановивши значення 2,1 як порогове.

2.1.2. Виключення середнього

Виключення середнього - методика попередньої обробки даних, що зазвичай використовується в машинному навчанні. Як правило, із векторів ознак (feature vectors) доцільно виключати середні значення, щоб кожна ознака (feature) центрувалася на нулі. Це робиться з метою, виключити з розгляду зміщення значень у векторах ознак.

2.1.3. Масштабування

У нашому векторі ознак кожне значення може змінюватись у деяких випадкових межах. Тому дуже важливо масштабувати ознаки, щоб вони були рівним ігровим полем для тренування алгоритму машинного навчання. Ми не хочемо, щоб будь-яка з ознак могла набувати штучно великого або малого значення лише через природу вимірів.

2.1.4. Нормалізація

Процес нормалізації полягає у зміні значень у векторі ознак таким чином, щоб для їх вимірювання можна було використовувати одну загальну шкалу. У машинному навчанні використовують різні форми нормалізації. У найбільш поширених з них, значення змінюються так, щоб їх сума дорівнювала 1. L1-нормалізація використовує метод найменших абсолютних відхилень (Least Absolute Deviations), що забезпечує рівність 1 суми абсолютних значень в кожному ряду. L2-нормалізація використовує метод найменших квадратів, що забезпечує рівність 1 суми квадратів значень. Взагалі, техніка L1-нормалізації вважається більш надійною по порівняно з L2-нормалізацією, оскільки вона менш чутлива до викидів.

Дуже часто дані містять викиди, і з цим нічого не вдієш. Ми хочемо використовувати безпечні методики, що дозволяють ігнорувати викиди у процесі обчислень. Якби ми вирішували завдання, в якому викиди грають важливу роль, то, ймовірно, найкращим вибором була б L2-нормалізація.

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([[5.1, -2.9, 3.3],
                       [-1.2, 7.8, -6.1],
                       [3.9, 0.4, 2.1],
                       [7.3, -9.9, -4.5]])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print("\n Binarized data:\n", data_binarized)

# Виведення середнього значення та стандартного відхилення
print("\nBEFORE: ")
```

```python
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))

# Исключение среднего
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))

# Масштабування MinMax
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\nMin max scaled data:\n", data_scaled_minmax)

# Нормалізація даних
data_normalized_l1 = preprocessing.normalize(input_data,
                                             norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data,
                                             norm='l2')
print("\nl1 normalized data:\n", data_normalized_l1)
print("\nl2 normalized data:\n", data_normalized_l2)
```

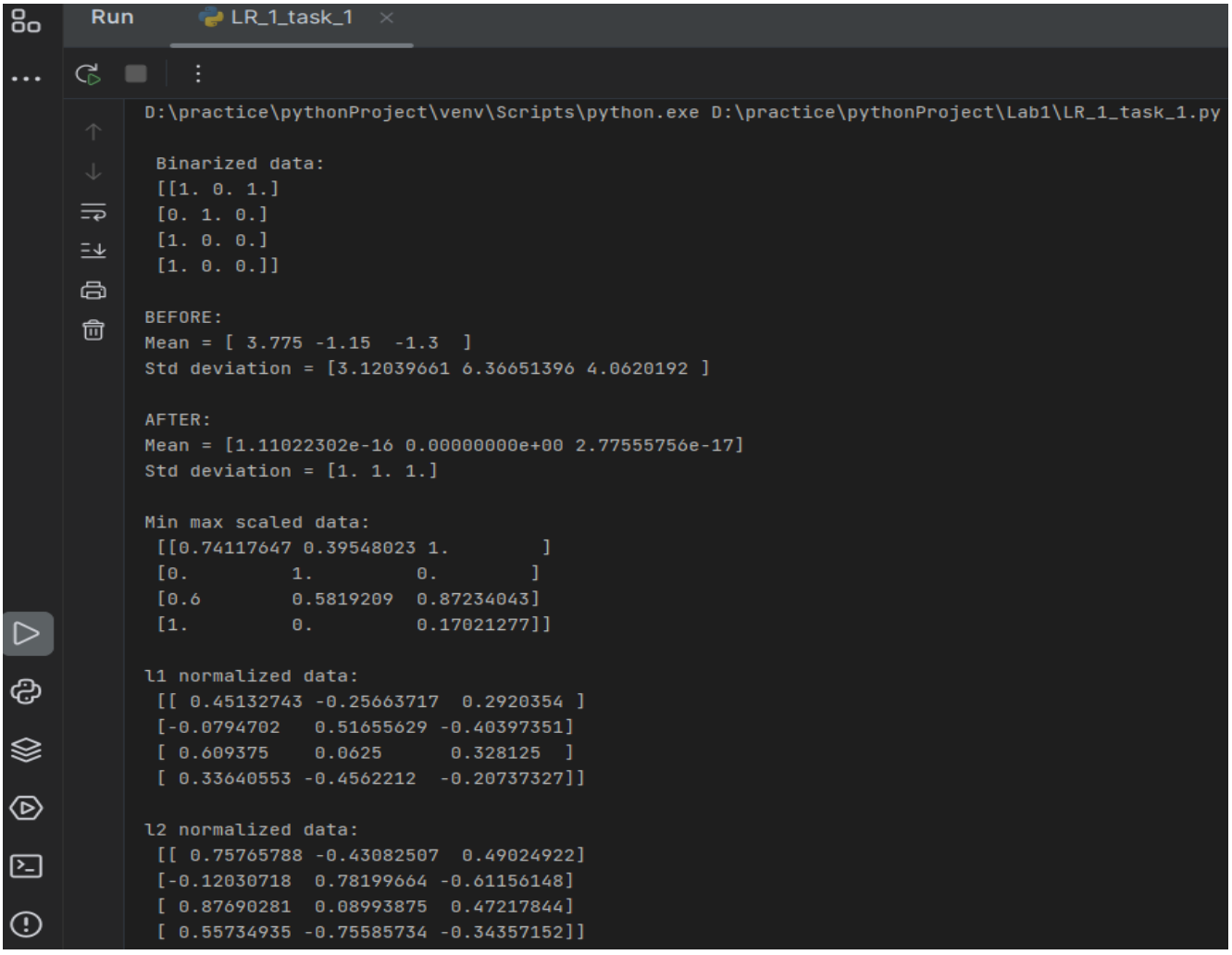


Рис. 1 Результат виконання програми

Різниця між L1 та L2 в тому, що у L1-нормалізації використовується сума абсолютних значень вагових значень, а L2-нормалізації - сума квадратів вагових значень, через це L1-нормалізація іноді дає побічний ефект видалення непотрібних функцій (features), присвоюючи пов'язаним з ними ваги значення 0.0, але не працює без проблем з усіма формами навчання, при цьому L2-нормалізація працює з усіма формами навчання, але не забезпечує неявної селекції функцій.

2.1.5. Кодування міток

Як правило, в процесі класифікації даних ми маємо справу з множиною міток (labels). Ними можуть бути слова, числа або інші об'єкти. Функції машинного навчання, що входять до бібліотеки sklearn, очікують, що мітки є числами. Тому, якщо мітки – це вже числа, ми можемо використовувати їх безпосередньо для того, щоб почати тренування. Однак, зазвичай, це не так. На практиці мітками служать слова, оскільки в такому вигляді вони краще всього сприймаються людиною. Ми позначаємо тренувальні дані словами, щоб полегшити відстеження відповідностей. Для перетворення слів у числа необхідно використовувати кодування. Під кодуванням міток (label encoding) мається на увазі процес перетворення словесних міток на числову форму. Завдяки цьому алгоритми можуть оперувати нашими даними.

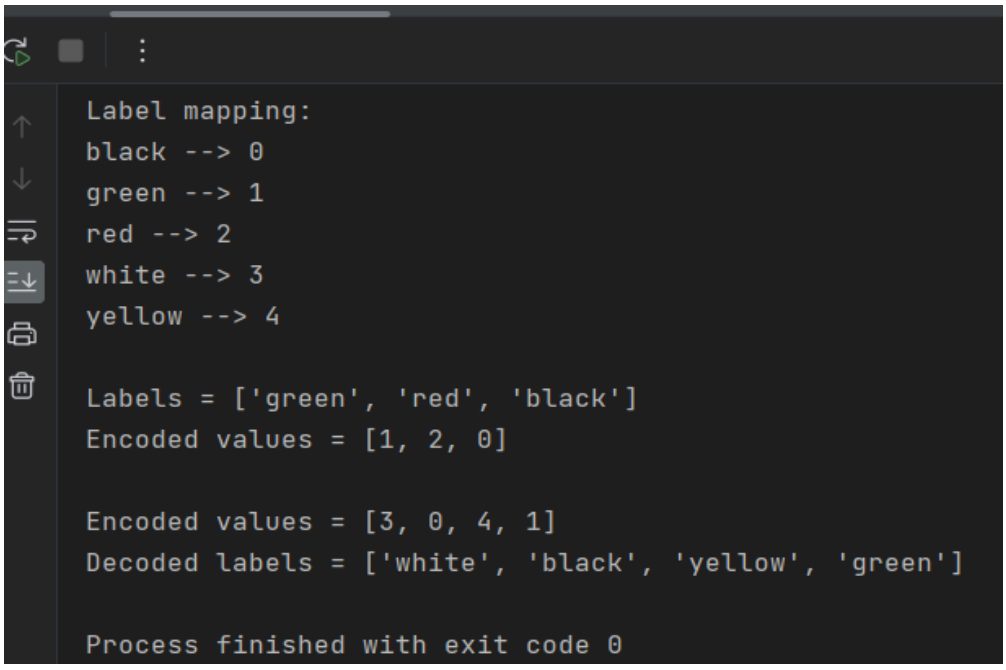


Рис. 2 Результат виконання програми

Завдання 2.2. Попередня обробка нових даних

У коді програми попереднього завдання поміняйте дані по рядках (значення змінної input_data) на значення відповідно варіанту таблиці 1 та виконайте операції: Бінарізації, Виключення середнього, Масштабування, Нормалізації.

Варіант обирається відповідно номера за списком групи відповідно до таблиці.

Варіант - 27

Значення змінної input_data:

4.6, 3.9, -3.5, -2.9, 4.1, 3.3, 2.2, 8.8, -4.1, 3.9, 2.4, 4.2

Поріг бінаризації: 2.2

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([[4.6, 3.9, -3.5],
                       [-2.9, 4.1, 3.3],
                       [2.2, 8.8, -4.1],
                       [3.9, 2.4, 4.2]])

data_binarized = preprocessing.Binarizer(threshold=2.2).transform(input_data)
print("\n Binarized data:\n", data_binarized)

print("\nBEFORE: ")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))

data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))

data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print("\nl1 normalized data:\n", data_normalized_l1)
print("\nl2 normalized data:\n", data_normalized_l2)
```

```
 Binarized data:
 [[1. 1. 0.]
 [0. 1. 1.]
 [0. 1. 0.]
 [1. 1. 1.]]

BEFORE:
Mean = [ 1.95   4.8   -0.025]
Std deviation = [2.93300188 2.40104144 3.79432141]

AFTER:
Mean = [-2.77555756e-17  1.11022302e-16  5.55111512e-17]
Std deviation = [1. 1. 1.]

l1 normalized data:
 [[ 0.38333333  0.325      -0.29166667]
 [-0.2815534   0.39805825  0.32038835]
 [ 0.14569536  0.58278146 -0.27152318]
 [ 0.37142857  0.22857143  0.4       ]]

l2 normalized data:
 [[ 0.65970588  0.55931585 -0.50195013]
 [-0.4825966   0.68229174  0.54916164]
 [ 0.22100788  0.88403154 -0.41187833]
 [ 0.62764591  0.38624364  0.67592637]]
```

Рис. 3 Результат виконання програми

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

Логістична регресія (logistic regression) - це методика, що використовується для пояснення відносин між вхідними та вихідними змінними. Вхідні змінні вважаються незалежними, вихідні – залежними. Залежна змінна може мати лише фіксований набір значень. Ці значення відповідають класам завдання класифікації. Метою є ідентифікація відносин між незалежними та залежними змінними за допомогою оцінки ймовірностей того, що та або інша залежна змінна відноситься до того чи іншого класу.

Лістинг програми:

```
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
from utilities import visualize_classifier

# Визначення зразка вхідних даних
X = np.array([[3.1, 7.2], [4, 6.7], [2.9, 8], [5.1, 4.5],
              [6, 5], [5.6, 5], [3.3, 0.4],
              [3.9, 0.9], [2.8, 1],
              [0.5, 3.4], [1, 4], [0.6, 4.9]])
y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])

# Створення логістичного класифікатора
classifier = linear_model.LogisticRegression(solver='liblinear', C=1)

# Тренування класифікатора
classifier.fit(X, y)

visualize_classifier(classifier, X, y)
```

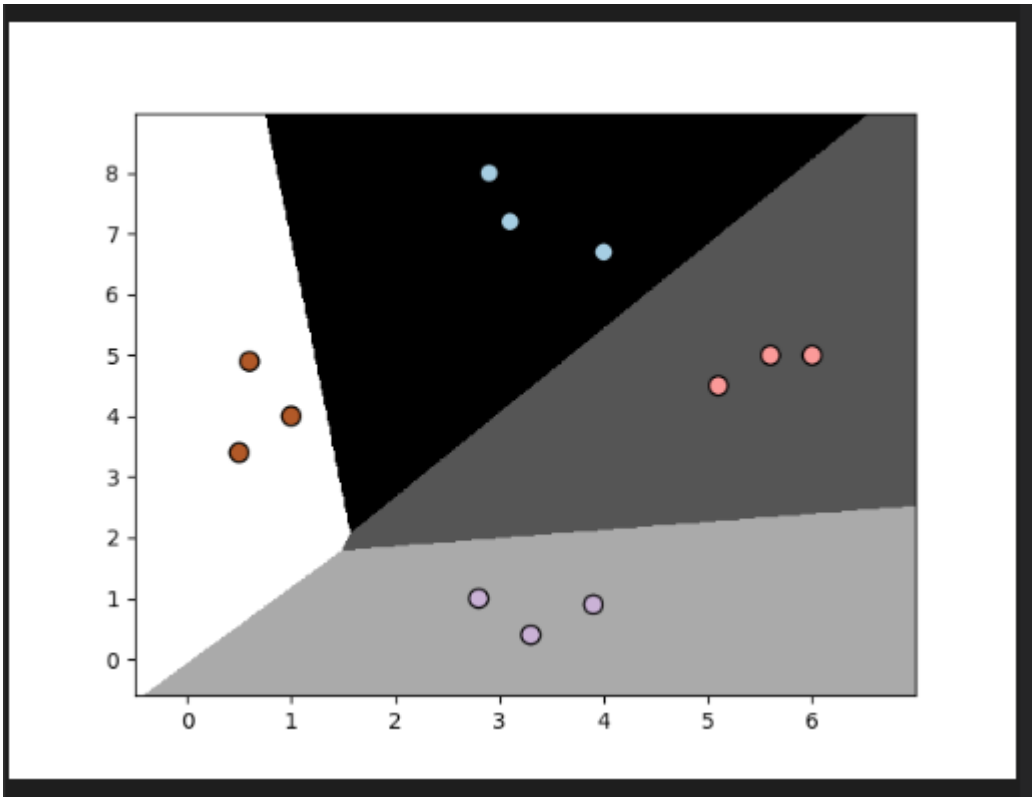


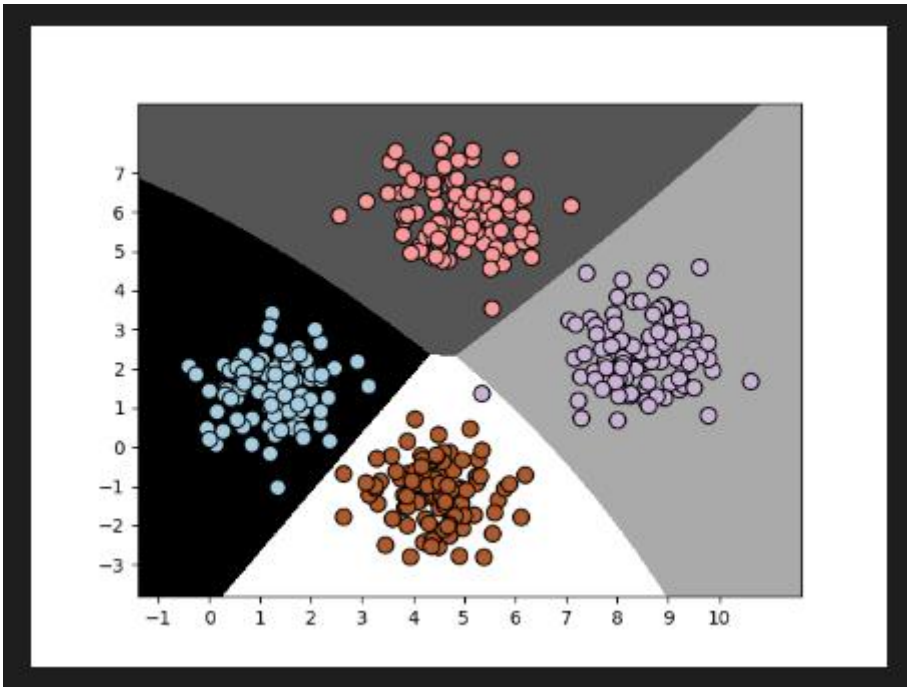
Рис. 4 Результат виконання програми

Завдання 2.4. Класифікація наївним байєсовським класифікатором

Наївний байєсовський :класифікатор (Na!ve Bayes classifier) - це простий класифікатор, заснований на використанні теореми Байєса, яка описує ймовірність події з урахуванням пов'язаних з нею умов. Такий класифікатор створюється за допомогою привласнення позначок класів екземплярам завдання. Останні представляються як векторів значень ознак. При цьому передбачається, що значення будь-якої заданої ознаки не залежить від значень інших ознак.

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from utilities import visualize_classifier

# Вхідний файл, який містить дані
input_file = 'data_multivar_nb.txt'
# Завантаження даних із вхідного файлу
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Створення наївного байєсовського класифікатора
classifier = GaussianNB()

# Тренування класифікатора
classifier.fit(X, y)

# Прогнозування значень для тренувальних даних
y_pred = classifier.predict(X)

# Обчислення якості класифікатора
accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")

# Візуалізація результатів роботи класифікатора
visualize_classifier(classifier, X, y)
```



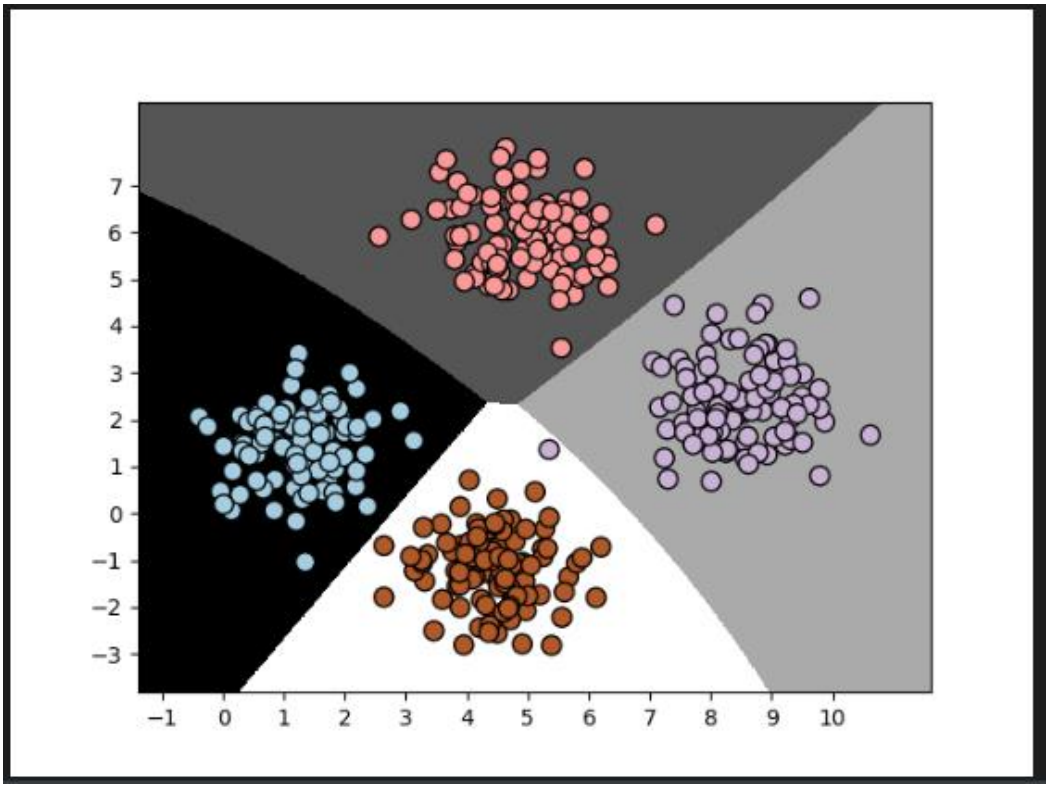

```
D:\practice\pythonProject\venv\Scripts\python.exe
Accuracy of Naive Bayes classifier = 99.75 %
```

Рис. 5 Результат виконання програми

Лістинг програми:

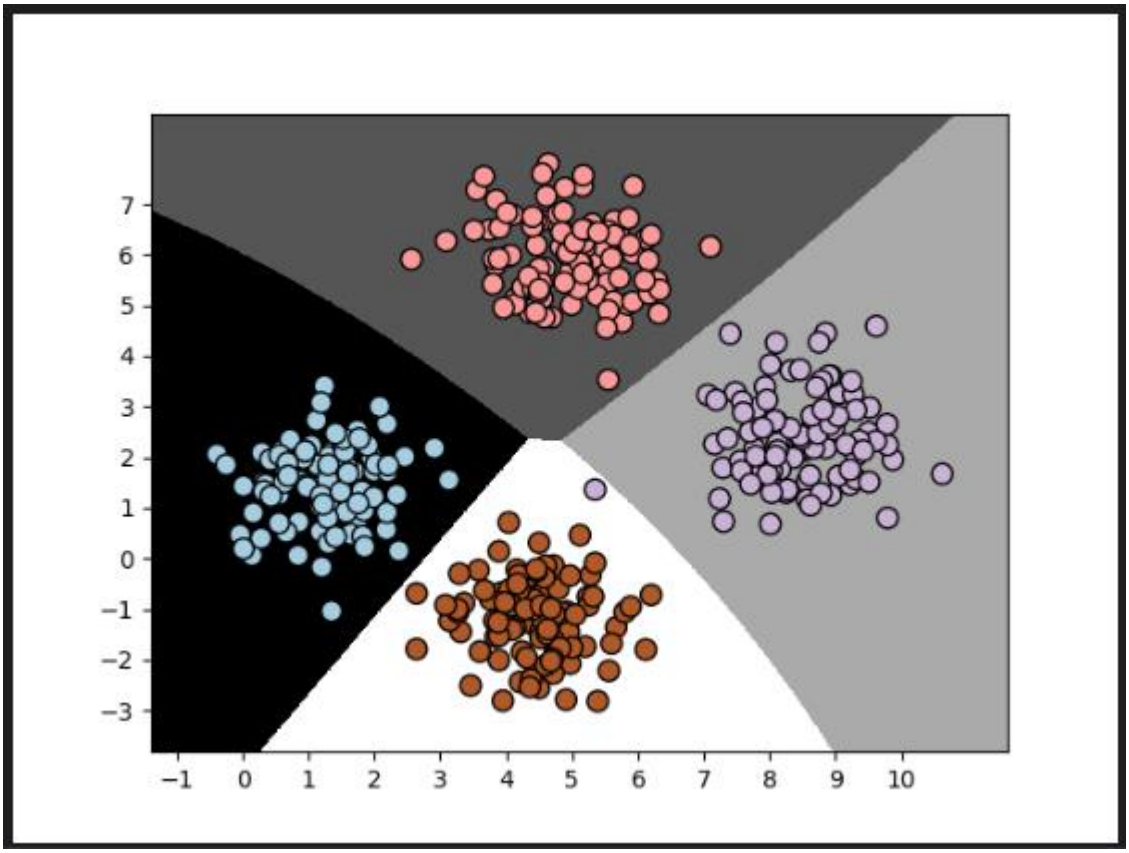
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split, cross_val_score
from utilities import visualize_classifier

# Вхідний файл, який містить дані
input_file = 'data_multivar_nb.txt'
# Завантаження даних із вхідного файлу
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Створення наївного байєсовського класифікатора
classifier = GaussianNB()

# Тренування класифікатора
classifier.fit(X, y)

# Прогнозування значень для тренувальних даних
y_pred = classifier.predict(X)

# Обчислення якості класифікатора
accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")

# Візуалізація результатів роботи класифікатора
```

```python
visualize_classifier(classifier, X, y)

# Continue task
num_folds = 3
accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy',
cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")

precision_values = cross_val_score(classifier, X, y, scoring='precision_weighted',
cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")

recall_values = cross_val_score(classifier, X, y, scoring='recall_weighted',
cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")

f1_values = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=num_folds)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")
```




```
D:\practice\pythonProject\venv\Scripts\python.exe
Accuracy of Naive Bayes classifier = 99.75 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Рис. 6 Результат виконання програми



```
D:\practice\pythonProject\venv\Scripts\python.exe
Accuracy of Naive Bayes classifier = 99.75 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Рис. 7 Результат виконання програми (другий прогін)

Висновок: програму запускав декілька разів і результат був такий самий як і перший раз. Тому таких вагомих від’ємностей я не побачив.

Завдання 2.5. Вивчити метрики якості класифікації

Лістинг програми:

```
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score, preci-
sion_score, f1_score, roc_curve, \
    roc_auc_score
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('data_metrics.csv')
df.head()

thresh = 0.5
df['predicted_RF'] = (df.model_RF >= 0.5).astype('int')
df['predicted_LR'] = (df.model_LR >= 0.5).astype('int')
df.head()

confusion_matrix(df.actual_label.values, df.predicted_RF.values)


def kovban_find_TP(y_true, y_pred):
    return sum((y_true == 1) & (y_pred == 1))


def kovban_find_FN(y_true, y_pred):
    return sum((y_true == 1) & (y_pred == 0))


def kovban_find_FP(y_true, y_pred):
    return sum((y_true == 0) & (y_pred == 1))


def kovban_find_TN(y_true, y_pred):
    return sum((y_true == 0) & (y_pred == 0))


print('TP:', kovban_find_TP(df.actual_label.values, df.predicted_RF.values))
print('FN:', kovban_find_FN(df.actual_label.values, df.predicted_RF.values))
print('FP:', kovban_find_FP(df.actual_label.values, df.predicted_RF.values))
print('TN:', kovban_find_TN(df.actual_label.values, df.predicted_RF.values))


def kovban_find_conf_matrix_values(y_true, y_pred):
    TP = kovban_find_TP(y_true, y_pred)
    FN = kovban_find_FN(y_true, y_pred)
    FP = kovban_find_FP(y_true, y_pred)
    TN = kovban_find_TN(y_true, y_pred)
    return TP, FN, FP, TN


def kovban_confusion_matrix(y_true, y_pred):
    TP, FN, FP, TN = kovban_find_conf_matrix_values(y_true, y_pred)
    return np.array([[TN, FP], [FN, TP]])


kovban_confusion_matrix(df.actual_label.values, df.predicted_RF.values)

assert np.array_equal(kovban_confusion_matrix(df.actual_label.values, df.pre-
dicted_RF.values),
                      confusion_matrix(df.actual_label.values, df.pre-
dicted_RF.values))
```

```
def kovban_accuracy_score(y_true, y_pred):
    TP, FN, FP, TN = kovban_find_conf_matrix_values(y_true, y_pred)
    return (TP + TN) / (TP + FP + TN + FN)


assert kovban_accuracy_score(df.actual_label.values, df.predicted_RF.values) ==
accuracy_score(df.actual_label.values,

df.predicted_RF.values), 'kovban_accuracy_score failed on RF'

assert kovban_accuracy_score(df.actual_label.values, df.predicted_LR.values) ==
accuracy_score(df.actual_label.values,

df.predicted_LR.values), 'kovban_accuracy_score failed on LR'

print('Accuracy RF: %.3f' % (kovban_accuracy_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('Accuracy LR: %.3f' % (kovban_accuracy_score(df.actual_label.values, df.pre-
dicted_LR.values)))

accuracy_score(df.actual_label.values, df.predicted_RF.values)
recall_score(df.actual_label.values, df.predicted_RF.values)


def kovban_recall_score(y_true, y_pred):
    TP, FN, FP, TN = kovban_find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FN)


assert kovban_recall_score(df.actual_label.values, df.predicted_RF.values) == re-
call_score(df.actual_label.values,

df.predicted_RF.values), 'kovban_recall_score failed on RF'

assert kovban_recall_score(df.actual_label.values, df.predicted_LR.values) == re-
call_score(df.actual_label.values,

df.predicted_LR.values), 'kovban_recall_score failed on LR'

print('Recall RF: %.3f' % (kovban_recall_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('Recall LR: %.3f' % (kovban_recall_score(df.actual_label.values, df.pre-
dicted_LR.values)))

precision_score(df.actual_label.values, df.predicted_RF.values)


def kovban_precision_score(y_true, y_pred):
    TP, FN, FP, TN = kovban_find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FP)


assert kovban_precision_score(df.actual_label.values, df.predicted_RF.values) ==
```

```
precision_score(
    df.actual_label.values, df.predicted_RF.values), 'kovban_precision_score
failed on RF'

assert kovban_precision_score(df.actual_label.values, df.predicted_LR.values) ==
precision_score(
    df.actual_label.values, df.predicted_LR.values), 'kovban_precision_score
failed on LR'

print('Precision RF: %.3f' % (kovban_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f' % (kovban_precision_score(df.actual_label.values,
df.predicted_LR.values)))

f1_score(df.actual_label.values, df.predicted_RF.values)


def kovban_f1_score(y_true, y_pred):
    TP, FN, FP, TN = kovban_find_conf_matrix_values(y_true, y_pred)
    recall = kovban_recall_score(y_true, y_pred)
    precision = kovban_precision_score(y_true, y_pred)
    return 2 * (recall * precision) / (recall + precision)


assert kovban_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values,

df.predicted_RF.values), 'kovban_f1_score failed on RF'
assert kovban_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values,

df.predicted_LR.values), 'kovban_f1_score failed on LR'

print('F1 RF: %.3f' % (kovban_f1_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('F1 LR: %.3f' % (kovban_f1_score(df.actual_label.values, df.pre-
dicted_LR.values)))

print('\nscores with threshold = 0.5')
print('Accuracy RF: %.3f' % (kovban_accuracy_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('Recall RF: %.3f' % (kovban_recall_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('Precision RF: %.3f' % (kovban_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f' % (kovban_f1_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('Accuracy LR: %.3f' % (kovban_accuracy_score(df.actual_label.values, df.pre-
dicted_LR.values)))
print('Recall LR: %.3f' % (kovban_recall_score(df.actual_label.values, df.pre-
dicted_LR.values)))
print('Precision LR: %.3f' % (kovban_precision_score(df.actual_label.values,
df.predicted_LR.values)))
```

```
print('F1 LR: %.3f' % (kovban_f1_score(df.actual_label.values, df.pre-
dicted_LR.values)))
print('')
print('scores with threshold = 0.25')
print(
    'Accuracy RF: %.3f' % (kovban_accuracy_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f' % (kovban_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f' % (
    kovban_precision_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('F1 RF: %.3f' % (kovban_f1_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print(
    'Accuracy LR: %.3f' % (kovban_accuracy_score(df.actual_label.values,
(df.model_LR >= 0.25).astype('int').values)))
print('Recall LR: %.3f' % (kovban_recall_score(df.actual_label.values,
(df.model_LR >= 0.25).astype('int').values)))
print('Precision LR: %.3f' % (
    kovban_precision_score(df.actual_label.values, (df.model_LR >=
0.25).astype('int').values)))
print('F1 LR: %.3f' % (kovban_f1_score(df.actual_label.values, (df.model_LR >=
0.25).astype('int').values)))

fpr_RF, tpr_RF, thresholds_RF = roc_curve(df.actual_label.values, df.model_RF.val-
ues)
fpr_LR, tpr_LR, thresholds_LR = roc_curve(df.actual_label.values, df.model_LR.val-
ues)

auc_RF = roc_auc_score(df.actual_label.values, df.model_RF.values)
auc_LR = roc_auc_score(df.actual_label.values, df.model_LR.values)
print('AUC RF:%.3f' % auc_RF)
print('AUC LR:%.3f' % auc_LR)

plt.plot(fpr_RF, tpr_RF, 'r-', label='RF AUC: %.3f' % auc_RF)
plt.plot(fpr_LR, tpr_LR, 'b-', label='LR AUC: %.3f' % auc_LR)
plt.plot([0, 1], [0, 1], 'k-', label='random')
plt.plot([0, 0, 1, 1], [0, 1, 1, 1], 'g-', label='perfect')
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

```
D:\practice\pythonProject\venv\Scripts\pyt
TP: 5047
FN: 2832
FP: 2360
TN: 5519
Accuracy RF: 0.671
Accuracy LR: 0.616
Recall RF: 0.641
Recall LR: 0.543
Precision RF: 0.681
Precision LR: 0.636
F1 RF: 0.660
F1 LR: 0.586

scores with threshold = 0.5
Accuracy RF: 0.671
Recall RF: 0.641
Precision RF: 0.681
F1 RF: 0.660
Accuracy LR: 0.616
Recall LR: 0.543
Precision LR: 0.636
F1 LR: 0.586

scores with threshold = 0.25
Accuracy RF: 0.502
Recall RF: 1.000
Precision RF: 0.501
F1 RF: 0.668
Accuracy LR: 0.503
Recall LR: 0.999
Precision LR: 0.501
F1 LR: 0.668
AUC RF:0.738
AUC LR:0.666
```

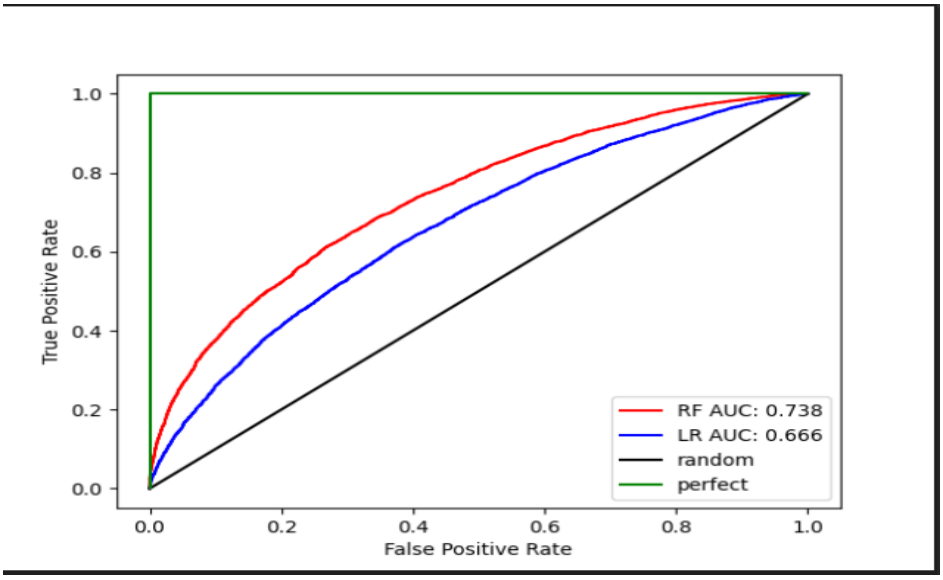


Рис. 8 Результат виконання програми

В даному завдані використовувались дві моделі – RF та LR. RF модель краще, так як є більш точною і повною, ніж LR. Це також можна побачити на графіку (рис. 8).

Завдання 2.6. Розробіть програму класифікації даних в файлі data_multivar_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVM). Розрахуйте показники якості класифікації. Порівняйте їх з показниками наївного байєсівського класифікатора. Зробіть висновки яку модель класифікації краще обрати і чому.

Лістинг програми:

```
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
from sklearn import preprocessing
from utilities import visualize_classifier

input_file = 'data_multivar_nb.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran-
dom_state=3)

classifier_svm = SVC()
classifier_svm.fit(X_train, y_train)

classifier_nb = GaussianNB()
classifier_nb.fit(X_train, y_train)

y_pred_svm = classifier_svm.predict(X_test)
y_pred_nb = classifier_nb.predict(X_test)

num_folds = 3

print('Naive Bayes:')
accuracy_values = cross_val_score(classifier_nb, X, y, scoring='accuracy',
cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2))
      + "%")

precision_values = cross_val_score(classifier_nb, X, y, scoring='preci-
sion_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(),
                                2)) + "%")

recall_values = cross_val_score(classifier_nb, X, y, scoring='recall_weighted',
cv=num_folds)
```

```
print("Recall: " + str(round(100 * recall_values.mean(), 2)) +
      "%")

f1_values = cross_val_score(classifier_nb, X, y, scoring='f1_weighted',
cv=num_folds)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")

print('\nSVM:')
accuracy_values = cross_val_score(classifier_svm, X, y, scoring='accuracy',
cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2))
      + "%")

precision_values = cross_val_score(classifier_svm, X, y, scoring='preci-
sion_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(),
                                2)) + "%")

recall_values = cross_val_score(classifier_svm, X, y, scoring='recall_weighted',
cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) +
      "%")

f1_values = cross_val_score(classifier_svm, X, y, scoring='f1_weighted',
cv=num_folds)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")

visualize_classifier(classifier_nb, X_test, y_test)
visualize_classifier(classifier_svm, X_test, y_test)
```

```
D:\practice\pythonProject\venv\Script
Naive Bayes:
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%

SVM:
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Рис. 9 Результат виконання програми

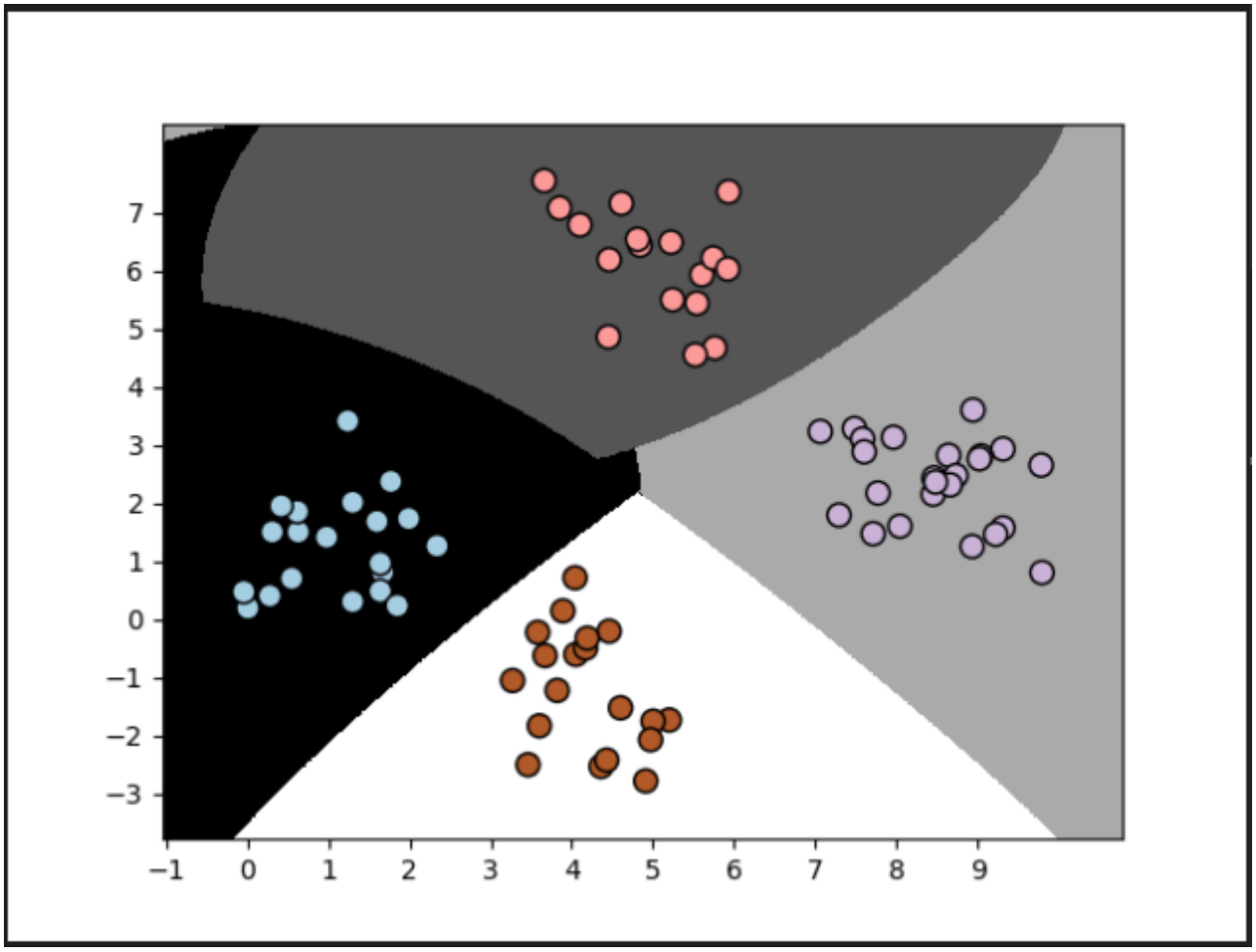


Рис. 10 Результат виконання програми

В даному випадку ми не має великої різниці між показниками SVM та показниками наївного байєсівського класифікатора, тому не модна визначити який з методів краще.

Висновки: на лабораторній роботі, я використовуючи спеціалізовані бібліотеки та мову програмування Python, дослідив попередню обробку та класифікацію даних.