ЛАБОРАТОРНА РОБОТА № 1
ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ
Варіант 17
Хід роботи:

Завдання 1:

Код програми:

```
import numpy as np
from sklearn import preprocessing


def binarize(target, threshold=2.1):
    binarizer = preprocessing.Binarizer(threshold=threshold)
    return binarizer.transform(target)


def print_mean_and_deviation(data):
    print(f"Mean: {data.mean(axis=0)}")
    print(f"Std deviation: {data.std(axis=0)}")


def scale(target):
    return preprocessing.scale(target)


def min_max_scale(target):
    scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
    return scaler.fit_transform(target)


def normalize(target, norm="l1"):
    return preprocessing.normalize(target, norm=norm)


input_data = np.array(
    [
        [1.3, 3.9, 6.2],
        [4.9, 2.2, -4.3],
        [-2.6, 6.5, 4.1],
        [-5.2, -3.4, -5.2],
    ]
)

binarized_data = binarize(input_data, 2.0)
print(f"Binarized data:\n{binarized_data}")

print()
```

```
print("BEFORE:")
print_mean_and_deviation(input_data)
scaled_data = scale(input_data)
print("AFTER:")
print_mean_and_deviation(scaled_data)

print()

min_max_scaled_data = min_max_scale(input_data)
print(f"Min max scaled data:\n{min_max_scaled_data}")

print()

l1_normalized_data = normalize(input_data, "l1")
print(f"l1 normalized data:\n{l1_normalized_data}")

print()

l2_normalized_data = normalize(input_data, "l2")
print(f"l2 normalized data:\n{l2_normalized_data}")
```

Виконання програми:

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

BEFORE:
Mean: [ 3.775 -1.15  -1.3  ]
Std deviation: [3.12039661 6.36651396 4.0620192 ]
AFTER:
Mean: [1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation: [1. 1. 1.]

Min max scaled data:
[[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]

l1 normalized data:
[[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.0794702   0.51655629 -0.40397351]
 [ 0.609375    0.0625      0.328125  ]
 [ 0.33640553 -0.4562212  -0.20737327]]

l2 normalized data:
[[ 0.75765788 -0.43082507  0.49024922]
 [-0.12030718  0.78199664 -0.61156148]
 [ 0.87690281  0.08993875  0.47217844]
 [ 0.55734935 -0.75585734 -0.34357152]]
PS B:\University\Системи штучного інтелекту>
```

Рис 1.1 – Результат виконання програми

Висновок: діапазон значень при L1 більший, коли як у L2 він менший, що змушує всі ознаки бути однаковими.

Завдання 2:
Код програми:

```
from sklearn import preprocessing


def train_encoder(target):
    encoder = preprocessing.LabelEncoder()
    encoder.fit(target)
    return encoder


def print_label_mapping(encoder):
    print("Label mapping:")
    for i, item in enumerate(encoder.classes_):
        print(f"{item} --> {i}")


def encode_labels(encoder, labels):
    return encoder.transform(labels)


def print_encoded_labels(input, output):
    print(f"Labels: {input}")
    print(f"Encoded values: {list(output)}")


def decode_values(encoder, values):
    return encoder.inverse_transform(values)


def print_decoded_values(input, output):
    print(f"Values: {input}")
    print(f"Decoded labels: {list(output)}")


input_labels = ["red", "black", "red", "green", "black", "yellow", "white"]

encoder = train_encoder(input_labels)
print_label_mapping(encoder)
print()

labels = ["green", "red", "black"]
encoded_values = encode_labels(encoder, labels)
print_encoded_labels(labels, encoded_values)
print()

values = [3, 0, 4, 1]
decoded_labels = decode_values(encoder, values)
```

```
print_decoded_values(values, decoded_labels)
```

Виконання програми:

```
black --> 0
green --> 1
red --> 2
white --> 3
yellow --> 4

Labels: ['green', 'red', 'black']
Encoded values: [np.int64(1), np.int64(2), np.int64(0)]

Values: [3, 0, 4, 1]
Decoded labels: [np.str_('white'), np.str_('black'), np.str_('yellow'), np.str_('green')]
PS B:\University\Системи штучного інтелекту>
```

Рис. 1.2 – Результат виконання програми

Завдання 3:

Код програми:

```
import numpy as np
from sklearn import preprocessing


def binarize(target, threshold=2.1):
    binarizer = preprocessing.Binarizer(threshold=threshold)
    return binarizer.transform(target)


def print_mean_and_deviation(data):
    print(f"Mean: {data.mean(axis=0)}")
    print(f"Std deviation: {data.std(axis=0)}")


def scale(target):
    return preprocessing.scale(target)


def min_max_scale(target):
    scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
    return scaler.fit_transform(target)


def normalize(target, norm="l1"):
    return preprocessing.normalize(target, norm=norm)


input_data = np.array(
    [
        [1.3, 3.9, 6.2],
        [4.9, 2.2, -4.3],
        [-2.6, 6.5, 4.1],
        [-5.2, -3.4, -5.2],
    ]
```

```
)

binarized_data = binarize(input_data, 2.0)
print(f"Binarized data:\n{binarized_data}")

print()

print("BEFORE:")
print_mean_and_deviation(input_data)
scaled_data = scale(input_data)
print("AFTER:")
print_mean_and_deviation(scaled_data)

print()

min_max_scaled_data = min_max_scale(input_data)
print(f"Min max scaled data:\n{min_max_scaled_data}")

print()

l1_normalized_data = normalize(input_data, "l1")
print(f"l1 normalized data:\n{l1_normalized_data}")

print()

l2_normalized_data = normalize(input_data, "l2")
print(f"l2 normalized data:\n{l2_normalized_data}")
```

Виконання програми:

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

BEFORE:
Mean: [-0.4  2.3  0.2]
Std deviation: [3.83601356 3.62973828 5.01547605]
AFTER:
Mean: [5.55111512e-17 5.55111512e-17 0.00000000e+00]
Std deviation: [1. 1. 1.]

Min max scaled data:
[[0.64356436 0.73737374 1.        ]
 [1.         0.56565657 0.07894737]
 [0.25742574 1.         0.81578947]
 [0.         0.         0.        ]]

l1 normalized data:
[[ 0.11403509  0.34210526  0.54385965]
 [ 0.42982456  0.19298246 -0.37719298]
 [-0.1969697   0.49242424  0.31060606]
 [-0.37681159 -0.24637681 -0.37681159]]

l2 normalized data:
[[ 0.17475265  0.52425796  0.83343572]
 [ 0.71216718  0.31974853 -0.62496303]
 [-0.32047519  0.80118797  0.50536472]
 [-0.64182859 -0.41965715 -0.64182859]]
PS B:\University\Системи штучного інтелекту>
```

Рис. 1.3 – Результат виконання програми

Завдання 4:

Код програми:

```python
import numpy as np
from sklearn import linear_model
from utilities import visualize_classifier


def create_classifier():
    return linear_model.LogisticRegression(solver="liblinear", C=1)


def train_classifier(classifier, x, y):
    classifier.fit(x, y)


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])

classifier = create_classifier()
train_classifier(classifier, x, y)
visualize_classifier(classifier, x, y)
```

Виконання програми:

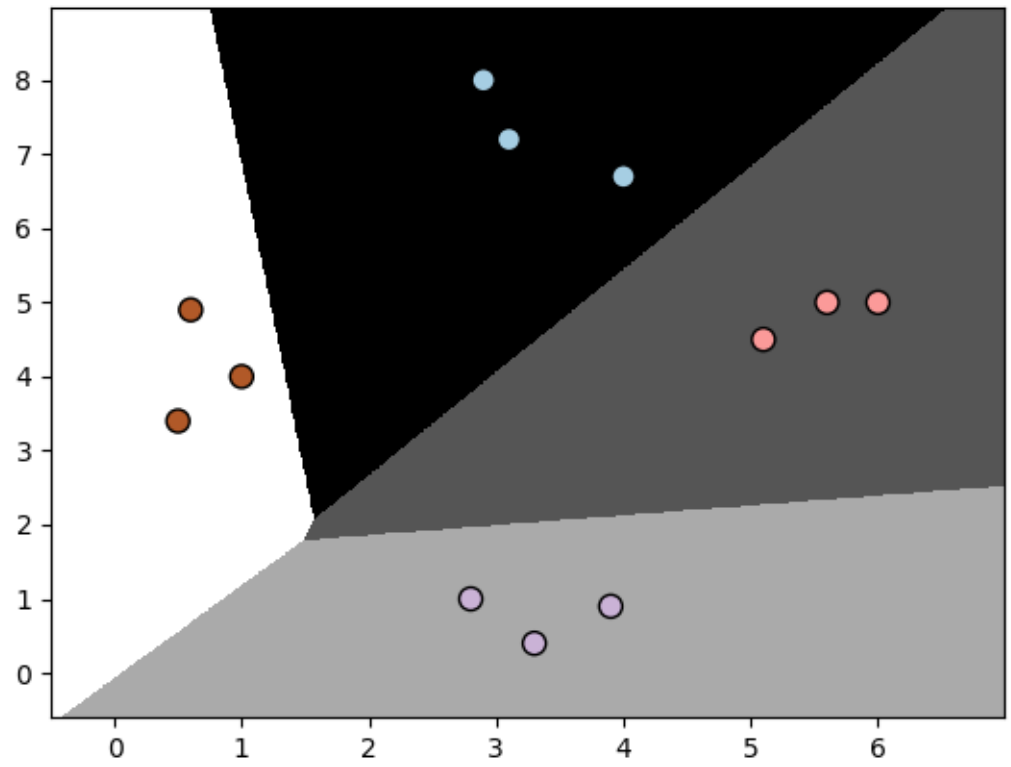

Рис. 1.4 – Результат виконання програми

Завдання 4:

Код програми:

```
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import GaussianNB

from utilities import visualize_classifier
```

```python
def load_arrays(filepath):
    data = np.loadtxt(filepath, delimiter=",")
    x = data[:, :-1]
    y = data[:, -1]
    return x, y


def train_classifier(x, y):
    classifier = GaussianNB()
    classifier.fit(x, y)
    return classifier


def get_prediction_accuracy(x, y, y_prediction):
    return 100.0 * (y == y_prediction).sum() / x.shape[0]


def print_accuracy(classifier, x, y, num_folds=3):
    values = cross_val_score(classifier, x, y, scoring="accuracy", cv=num_folds)
    print("Accuracy: " + str(round(100 * values.mean(), 2)) + "%")


def print_precision(classifier, x, y, num_folds=3):
    values = cross_val_score(
        classifier, x, y, scoring="precision_weighted", cv=num_folds
    )
    print("Precision: " + str(round(100 * values.mean(), 2)) + "%")


def print_recall(classifier, x, y, num_folds=3):
    values = cross_val_score(classifier, x, y, scoring="recall_weighted",
cv=num_folds)
    print("Recall: " + str(round(100 * values.mean(), 2)) + "%")


def print_f1(classifier, x, y, num_folds=3):
    values = cross_val_score(classifier, x, y, scoring="f1_weighted", cv=num_folds)
    print("F1: " + str(round(100 * values.mean(), 2)) + "%")


x, y = load_arrays("Лабораторна робота 1/data_multivar_nb.txt")
classifier = train_classifier(x, y)

y_prediction = classifier.predict(x)
accuracy = get_prediction_accuracy(x, y, y_prediction)
print(f"Accuracy of Naive Bayes = {round(accuracy, 2)}%")
print_accuracy(classifier, x, y)
print_precision(classifier, x, y)
print_recall(classifier, x, y)
```

```
print_f1(classifier, x, y)


x_training, x_test, y_training, y_test = train_test_split(
    x, y, test_size=0.2, random_state=3
)
new_classifier = train_classifier(x_training, y_training)

y_test_prediction = new_classifier.predict(x_test)
accuracy = get_prediction_accuracy(x_test, y_test, y_test_prediction)
print(f"Accuracy of the new classifier = {round(accuracy, 2)}%")

visualize_classifier(new_classifier, x_test, y_test)
```

Виконання програми:

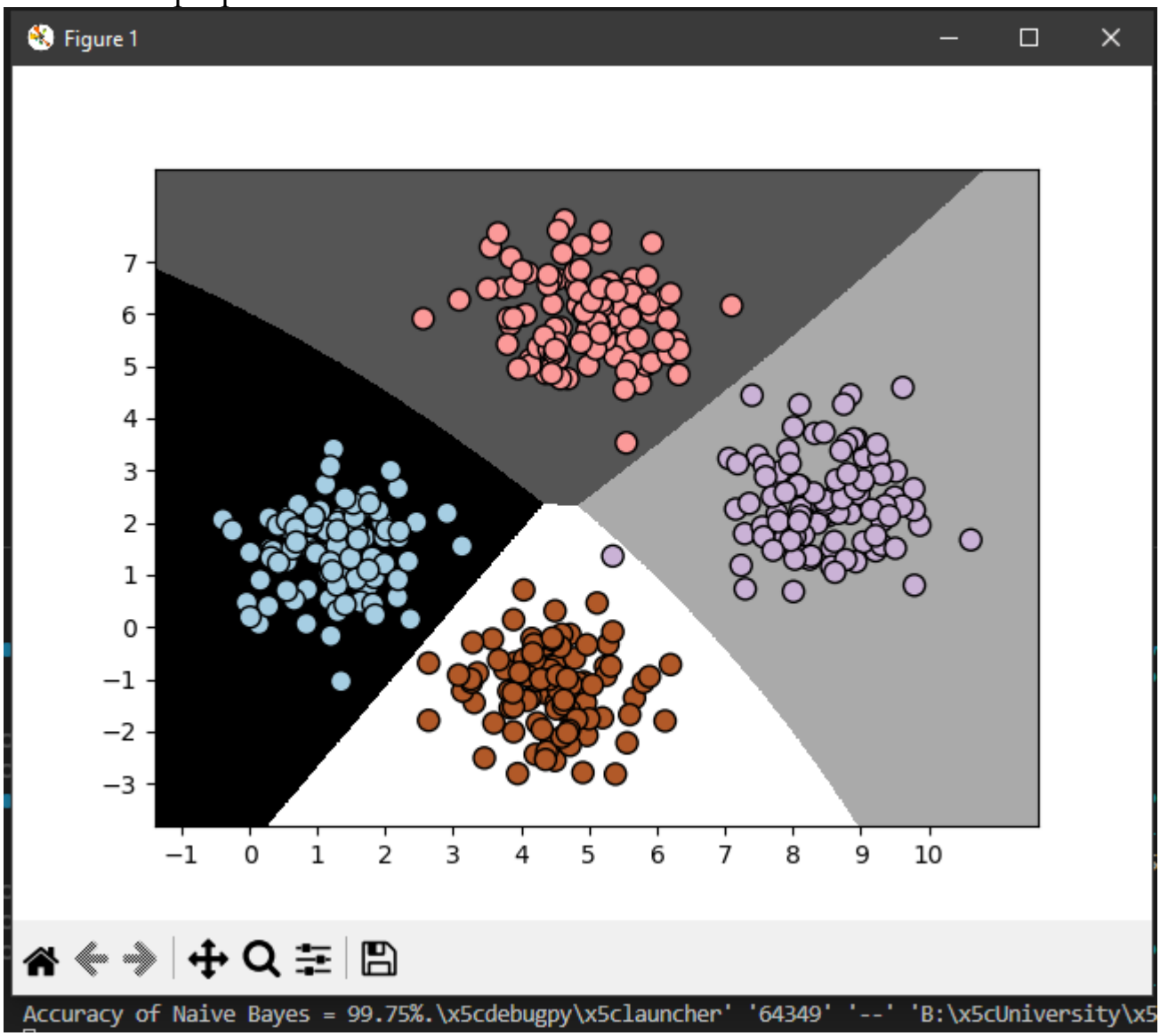

Рис. 1.5 – Результат виконання програми

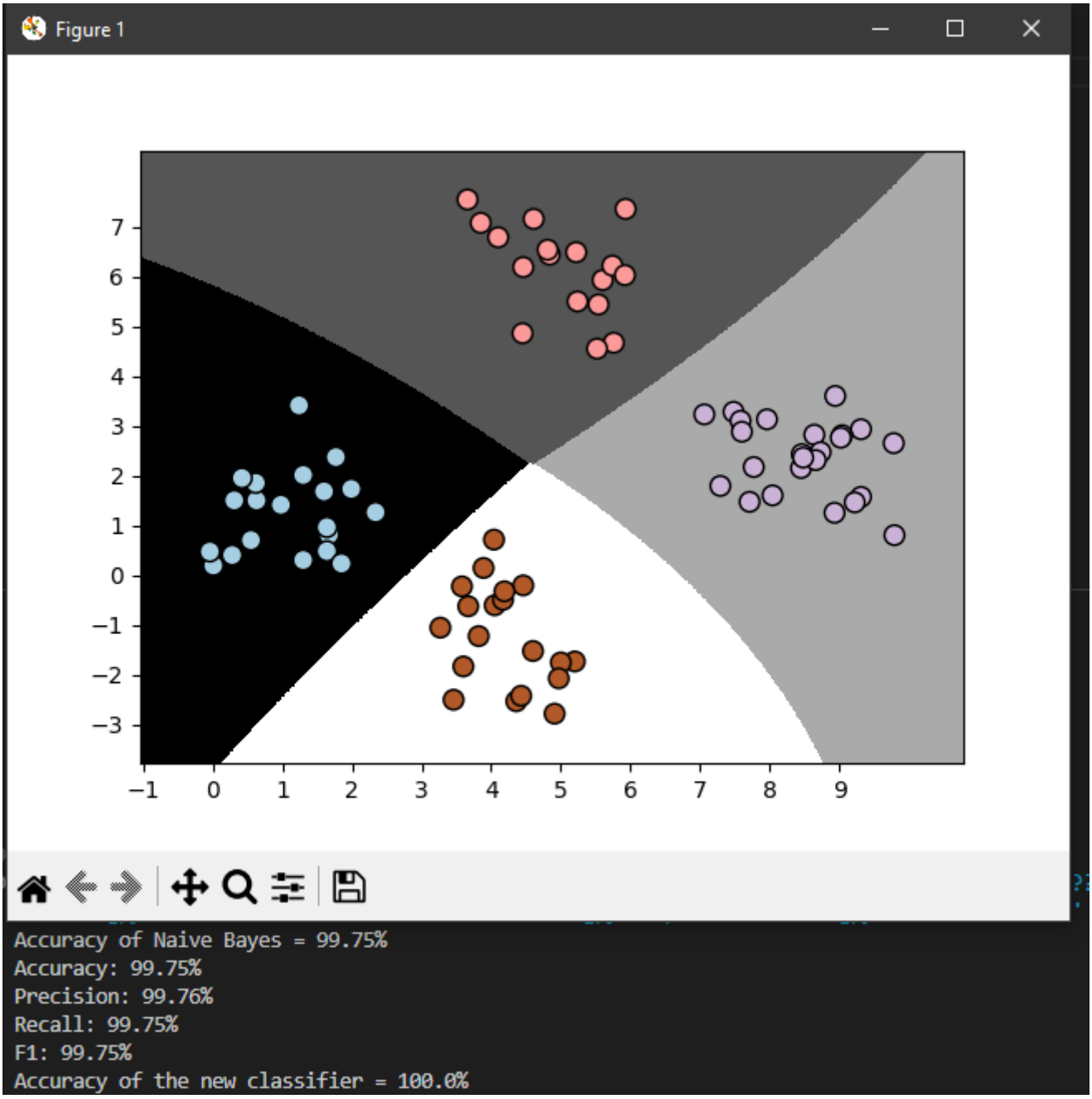

Рис. 1.6 – Перше виконання програми

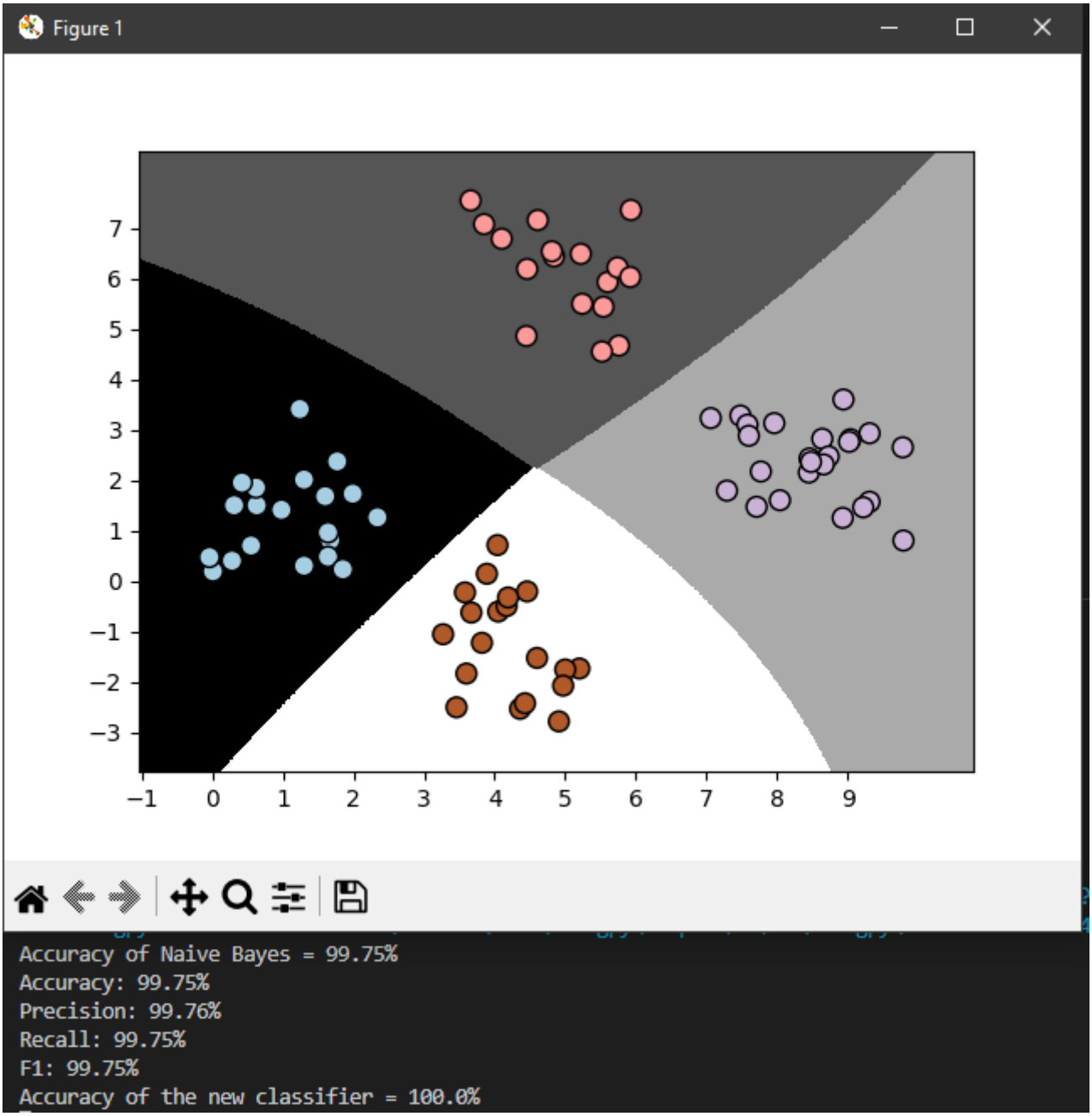

Рис. 1.7 – Друге виконання програми

Висновок: Перший та другий прогін програми нічим не відрізняються

Завдання 5:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix as get_confusion_matrix
from sklearn.metrics import accuracy_score as get_accuracy_score
from sklearn.metrics import recall_score as get_recall_score
from sklearn.metrics import precision_score as get_precision_score
from sklearn.metrics import f1_score as get_f1_score
from sklearn.metrics import roc_curve as get_roc_curve
from sklearn.metrics import roc_auc_score as get_roc_auc_score
```

```python
def get_dataframe(filename):
    dataframe = pd.read_csv(filename)

    return dataframe


def binarize(y, threshold=0.5):
    y_prediction = np.where(y >= threshold, 1, 0)
    return y_prediction


def korniichuk_confusion_matrix(y, y_prediction):
    tp = korniichuk_tp(y, y_prediction)
    fn = korniichuk_fn(y, y_prediction)
    fp = korniichuk_fp(y, y_prediction)
    tn = korniichuk_tn(y, y_prediction)
    return np.array([[tn, fp], [fn, tp]])


def korniichuk_tp(y, y_prediction):
    result = 0
    for actual, predicted in zip(y, y_prediction):
        if actual == 1 and predicted == 1:
            result += 1
    return result


def korniichuk_fn(y, y_prediction):
    result = 0
    for actual, predicted in zip(y, y_prediction):
        if actual == 1 and predicted == 0:
            result += 1
    return result


def korniichuk_fp(y, y_prediction):
    result = 0
    for actual, predicted in zip(y, y_prediction):
        if actual == 0 and predicted == 1:
            result += 1
    return result


def korniichuk_tn(y, y_prediction):
    result = 0
    for actual, predicted in zip(y, y_prediction):
        if actual == 0 and predicted == 0:
            result += 1
    return result
```

```python
def split_confusion_matrix(matrix):
    tn = matrix[0][0]
    fp = matrix[0][1]
    fn = matrix[1][0]
    tp = matrix[1][1]
    return tp, fn, fp, tn


def print_confusion_matrix(matrix):
    tp, fn, fp, tn = split_confusion_matrix(matrix)
    print(f"TP: {tp}")
    print(f"FN: {fn}")
    print(f"FP: {fp}")
    print(f"TN: {tn}")


def korniichuk_accuracy_score(confusion_matrix):
    tp, fn, fp, tn = split_confusion_matrix(confusion_matrix)
    return (tp + tn) / (tp + tn + fp + fn)


def korniichuk_recall_score(confusion_matrix):
    tp, fn, _, _ = split_confusion_matrix(confusion_matrix)
    return tp / (tp + fn)


def korniichuk_precision_score(confusion_matrix):
    tp, _, fp, _ = split_confusion_matrix(confusion_matrix)
    return tp / (tp + fp)


def korniichuk_f1_score(confusion_matrix):
    recall_score = korniichuk_recall_score(confusion_matrix)
    precision_score = korniichuk_precision_score(confusion_matrix)
    return (2 * (precision_score * recall_score)) / (precision_score + recall_score)


def build_roc_plot(fpr_y1, tpr_y1, fpr_y2, tpr_y2):
    plt.plot(fpr_y1, tpr_y1, "r-", label="RF")
    plt.plot(fpr_y2, tpr_y2, "b-", label="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()


dataframe = get_dataframe("Лабораторна робота 1/data_metrics.csv")

y, y1_prediction, y2_prediction = (
```

```
    dataframe.actual_label.array,
    dataframe.model_RF.array,
    dataframe.model_LR.array,
)
y1_prediction = binarize(y1_prediction)

# CONFUSION_MATRIX
my_confusion_matrix = korniichuk_confusion_matrix(
    y,
    y1_prediction,
)
confusion_matrix = get_confusion_matrix(y, y1_prediction)

print("Korniichuk Confusion matrix:")
print_confusion_matrix(my_confusion_matrix)

print("\nSKLearn Confusion matrix:")
print_confusion_matrix(confusion_matrix)

assert np.array_equal(confusion_matrix, my_confusion_matrix)

# ACCURACY_SCORE
my_accuracy_score = korniichuk_accuracy_score(confusion_matrix)
print(f"\nKorniichuk accuracy score = {my_accuracy_score}")

accuracy_score = get_accuracy_score(y, y1_prediction)
print(f"SKLearn accuracy score = {accuracy_score}")

assert my_accuracy_score == accuracy_score

# RECALL_SCORE
my_recall_score = korniichuk_recall_score(confusion_matrix)
print(f"\nKorniichuk recall score = {my_recall_score}")

recall_score = get_recall_score(y, y1_prediction)
print(f"SKLearn recall score = {recall_score}")

assert my_recall_score == recall_score

# PRECISION_SCORE
my_precision_score = korniichuk_precision_score(confusion_matrix)
print(f"\nKorniichuk precision score = {my_precision_score}")

precision_score = get_precision_score(y, y1_prediction)
print(f"SKLearn precision score = {precision_score}")

assert my_precision_score == precision_score

# F1_SCORE
my_f1_score = korniichuk_f1_score(confusion_matrix)
```

```
print(f"\nKorniichuk f1 score = {my_f1_score}")

f1_score = get_f1_score(y, y1_prediction)
print(f"SKLearn f1 score = {f1_score}")

assert my_f1_score == f1_score

# ROC_CURVE
fpr_y1, tpr_y1, thresholds_y1 = get_roc_curve(y, y1_prediction)
fpr_y2, tpr_y2, thresholds_y2 = get_roc_curve(y, y2_prediction)
build_roc_plot(fpr_y1, tpr_y1, fpr_y2, tpr_y2)


# ROC_AUC_SCORE
roc_auc_score1 = get_roc_auc_score(y, y1_prediction)
print(f"\nAUC 1 = {roc_auc_score1}")
roc_auc_score2 = get_roc_auc_score(y, y2_prediction)
print(f"AUC 2 = {roc_auc_score2}")
```

Виконання програми:

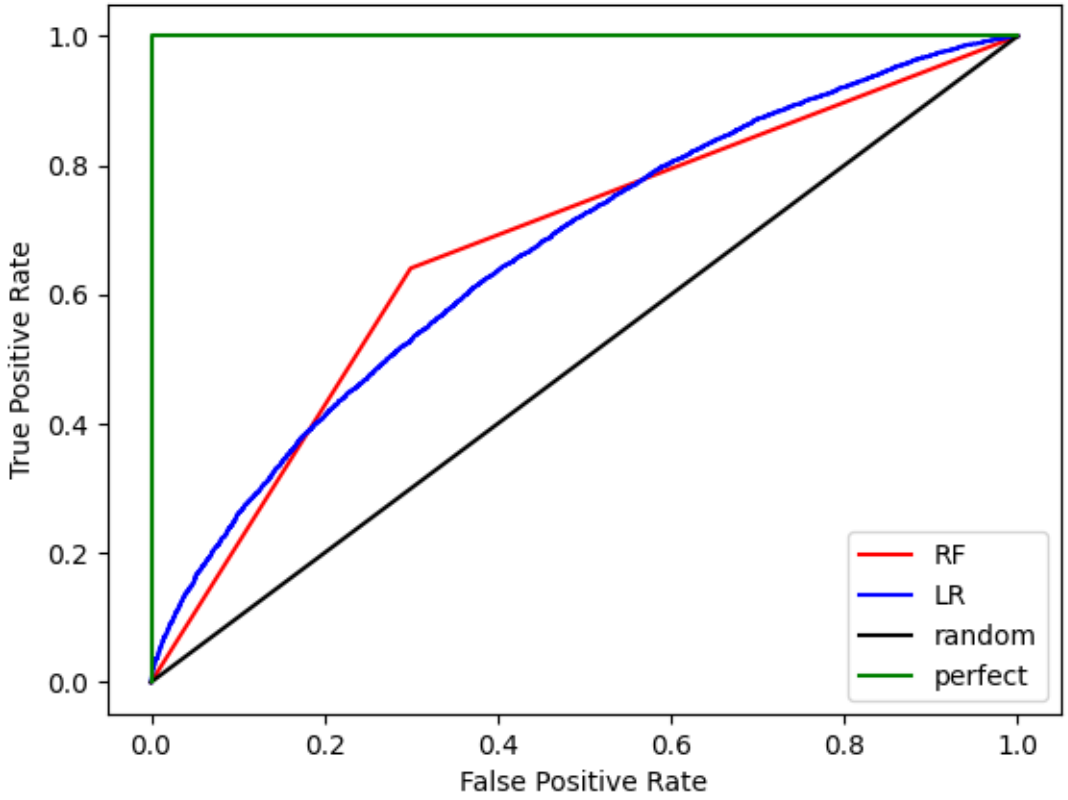

Рис. 1.8 – Виконання програми

```
Korniichuk Confusion matrix:
TP: 5047
FN: 2832
FP: 2360
TN: 5519

SKLearn Confusion matrix:
TP: 5047
FN: 2832
FP: 2360
TN: 5519

Korniichuk accuracy score = 0.6705165630156111
SKLearn accuracy score = 0.6705165630156111

Korniichuk recall score = 0.6405635232897576
SKLearn recall score = 0.6405635232897576

Korniichuk precision score = 0.681382476036182
SKLearn precision score = 0.681382476036182

Korniichuk f1 score = 0.660342797330891
SKLearn f1 score = 0.660342797330891

AUC 1 = 0.670516563015611
AUC 2 = 0.6657435203840882
```

Рис. 1.9 – Виконання програми

Висновки: RF модель є кращою, тому що її roc auc вище

Завдання 6:

Код програми:

```
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix as get_confusion_matrix


def load_arrays(filepath):
    data = np.loadtxt(filepath, delimiter=",")
    x = data[:, :-1]
    y = data[:, -1]
    return x, y


def create_svm(x, y):
    svm = SVC()
    svm.fit(x, y)
    return svm


def create_naive(x, y):
    naive = GaussianNB()
```

```
    naive.fit(x, y)
    return naive


def predict(classifier, x):
    return classifier.predict(x)


def evaluate_model(classifier, x, y, num_folds=3):
    accuracy = cross_val_score(
        classifier, x, y, cv=num_folds, scoring="accuracy"
    ).mean()
    precision = cross_val_score(
        classifier, x, y, cv=num_folds, scoring="precision_macro"
    ).mean()

    recall = cross_val_score(
        classifier, x, y, cv=num_folds, scoring="recall_macro"
    ).mean()
    f1 = cross_val_score(classifier, x, y, cv=num_folds, scoring="f1_macro").mean()

    print(f"Accuracy = {accuracy}")
    print(f"Precision = {precision}")
    print(f"Recall = {recall}")
    print(f"F1-score = {f1}")

    y_prediction = classifier.predict(x)
    confusion_matrix = get_confusion_matrix(y, y_prediction)
    print("Confusion Matrix:\n", confusion_matrix)


x, y = load_arrays("Лабораторна робота 1/data_multivar_nb.txt")
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, ran-
dom_state=3)

svm = create_svm(x_train, y_train)
naive = create_naive(x_train, y_train)

svm_prediction = predict(svm, x_test)
naive_prediction = predict(naive, x_test)

print("SVM:")
evaluate_model(svm, x_test, y_test)

print("\nNaive Bayes:")
evaluate_model(naive, x_test, y_test)
```

Виконання програми:

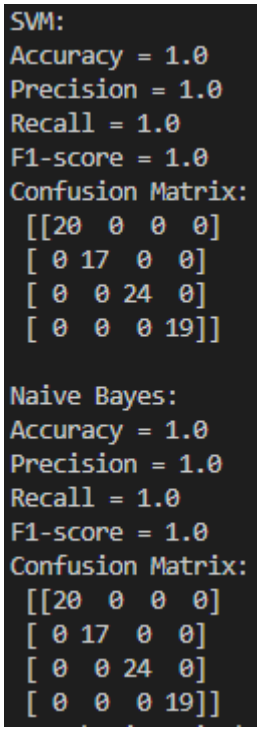

Рис. 1.10 – Виконання програми

Посилання на GitHub: https://github.com/ipz215kvv/artificial-inteligence-systems