Лабораторна робота № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

Завдання 2.1. Попередня обробка даних

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

```
# Виключення середнього
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)
```

Результат 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]]
```

Рисунок 1 — Результат виконання програми.

Чим відрізняється L1-нормалізація від L2-нормалізації?

Нормалізація L1 робить абсолютні значення кожного рядка рівними 1, роблячи дані менш чутливими до великих значень. L2-нормалізація перетворює дані так, що сума квадратів значень у кожному рядку дорівнює 1, що зменшує вплив екстремальних значень і зберігає геометричну структуру даних.

Завдання 2.1.5. Кодування міток

Програмний код:

```
import numpy as np
```

```
from sklearn import preprocessing

# Надання позначок вхідних даних
Input_labels = ['red', 'Ыack', 'red', 'green', 'Ьlack', 'yellow', 'white']

# Створення кодувальника та встановлення відповідності
# між мітками та числами
encoder = preprocessing.LabelEncoder()
encoder.fit(Input_labels)

# Виведення відображення
print("\nLabel mapping:")
for i, item in enumerate(encoder.classes_):
    print(item, '-->', i)

# перетворення міток за допомогою кодувальника
test_labels = ['green', 'red', 'Ыack']
encoded_values = encoder.transform(test_labels)
print("\nLabels =", test_labels)
print("Encoded values =", list(encoded_values))

# Декодування набору чисел за допомогою декодера
encoded_values = [3, 0, 4, 1]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded_values)
print("Decoded labels =", list(decoded_list))
```

Результат виконання програми:

```
Label mapping:
green --> 0
red --> 1
white --> 2
yellow --> 3
Black --> 4
black --> 5

Labels = ['green', 'red', 'Black']
Encoded values = [0, 1, 4]

Encoded values = [3, 0, 4, 1]
Decoded labels = ['yellow', 'green', 'Black', 'red']
```

Рисунок 2 — Результат виконання програми.

Завдання 2.2. Попередня обробка нових даних

Мій варіант №6

6.	2.3	-1.6	6.1	-2.4	-1.2	4.3	3.2	5.5	-6.1	-4.4	1.4	-1.2	2.1

Код програми:

```
import numpy as np
from sklearn import preprocessing


input_data = np.array([[2.3, -1.6, 6.1],
                       [-2.4, -1.2, 4.3],
                       [3.2, 5.5, -6.1],
                       [-4.4, 1.4, -1.2]])


# 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: ")
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)
```

Результат виконання програми:

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

BEFORE:
Mean = [-0.325  1.025  0.775]
Std deviation = [3.17125764 2.82875856 4.79446295]

AFTER:
Mean = [ 5.55111512e-17 -5.55111512e-17  6.93889390e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.88157895 0.         1.        ]
 [0.26315789 0.05633803 0.85245902]
 [1.         1.         0.        ]
 [0.         0.42253521 0.40163934]]

l1 normalized data:
 [[ 0.23       -0.16        0.61      ]
 [-0.30379747 -0.15189873  0.5443038 ]
 [ 0.21621622  0.37162162 -0.41216216]
 [-0.62857143  0.2        -0.17142857]]

l2 normalized data:
 [[ 0.34263541 -0.23835507  0.90872869]
 [-0.47351004 -0.23675502  0.84837215]
 [ 0.36302745  0.62395344 -0.69202108]
```

Рисунок 3 — Результат виконання програми.

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор.

Код програми:

```
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# 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])

def visualize_classifier(classifier, X, y):
    # Визначаємо межі графіку
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    # Передбачення класу для кожної точки сітки
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Побудова графіку
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(('red', 'blue', 'green',
'yellow')))
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k',
```

```
cmap=ListedColormap(('red', 'blue', 'green', 'yellow')))
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.show()


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

# Тренування класифікатора
classifier.fit(X, y)

visualize_classifier(classifier, X, y)
```

Результат виконання програми:

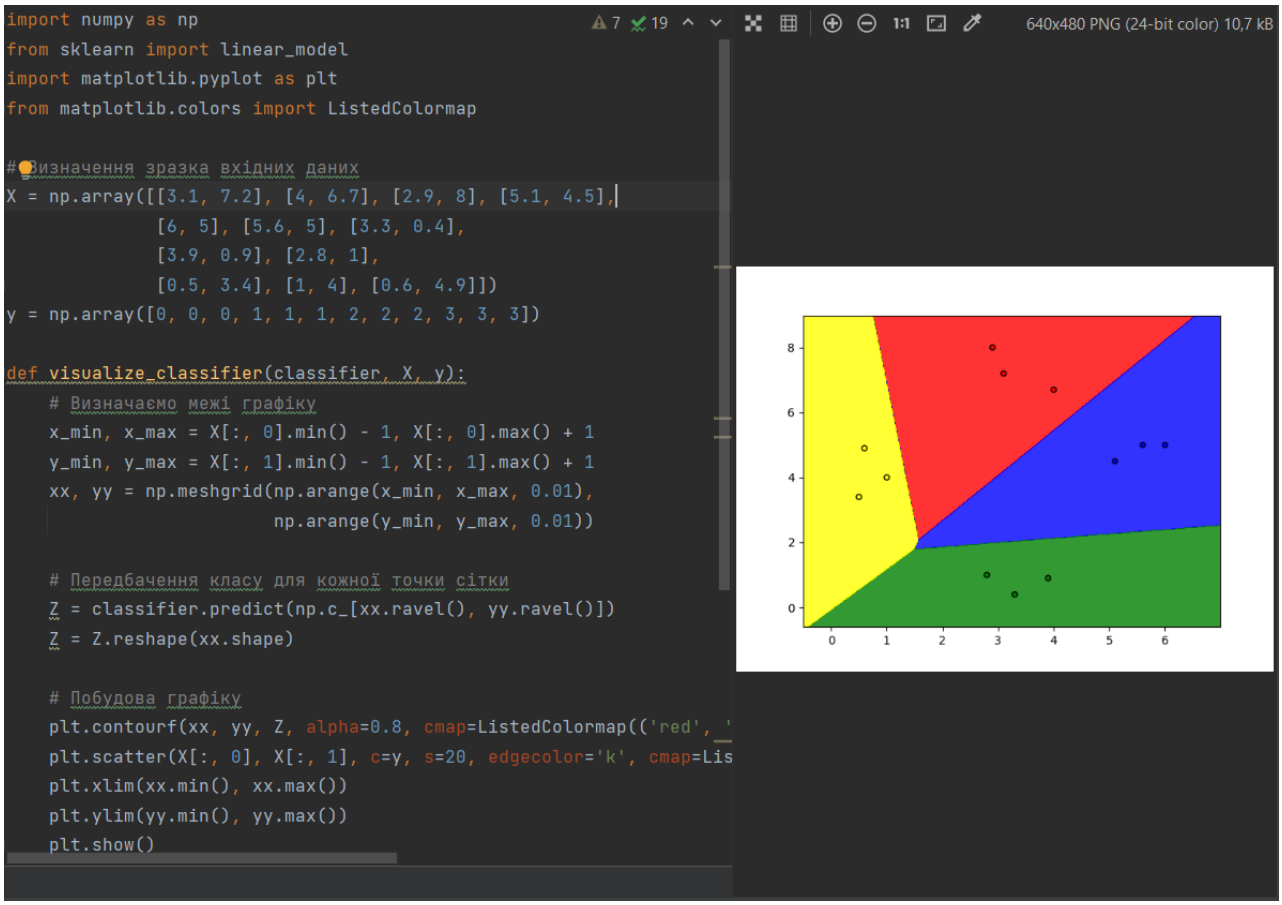


Рисунок 4 — Результат виконання програми.

Завдання 2.4. Класифікація наївним байєсовським класифікатором

Код програми:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split



def visualize_classifier(classifier, X, y):
    # Визначаємо межі графіку
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    # Передбачення класу для кожної точки сітки
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Побудова графіку
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(('red', 'blue', 'green',
'yellow')))
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k',
cmap=ListedColormap(('red', 'blue', 'green', 'yellow')))
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.show()
```

```
# Вхідний файл, який містить дані
input_file = '../data/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)
```

Результат виконання програми:

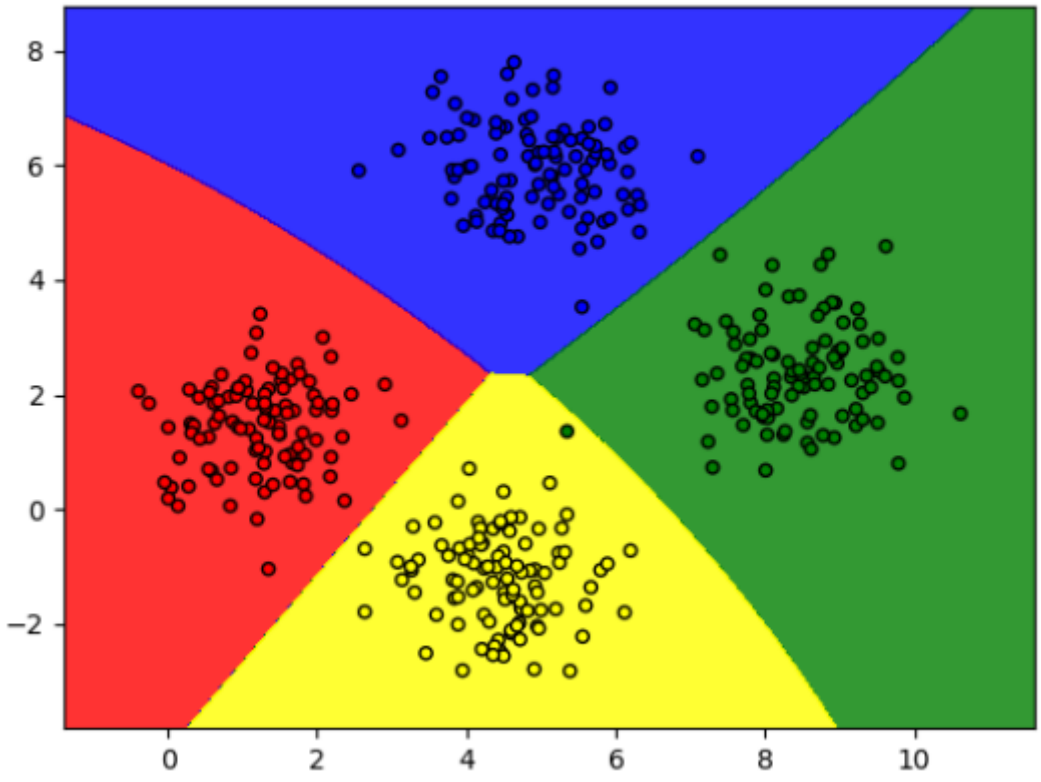


Рисунок 5 — Результат виконання програми.

Код апгрейд версії:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split, cross_val_score,
StratifiedKFold
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, \
    ConfusionMatrixDisplay
from collections import Counter


def visualize_classifier(classifier, X, y):
    # Визначаємо межі графіку
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    # Передбачення класу для кожної точки сітки
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Побудова графіку
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(('red', 'blue', 'green',
'yellow')))
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k',
cmap=ListedColormap(('red', 'blue', 'green', 'yellow')))
```

```
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.show()


# Вхідний файл, який містить дані
input_file = '../data/data_multivar_nb.txt'

# Завантаження даних із вхідного файлу
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Розбивка 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=3)

# Створення та тренування нового наївного байєсовського класифікатора
classifier_new = GaussianNB()
classifier_new.fit(X_train, y_train)

# Прогнозування значень для тестових даних
y_test_pred = classifier_new.predict(X_test)

# Обчислення якості класифікатора на тестових даних
accuracy = accuracy_score(y_test, y_test_pred)
print("Accuracy of the new classifier =", round(accuracy * 100, 2), "%")

# Візуалізація роботи класифікатора на тестових даних
visualize_classifier(classifier_new, X_test, y_test)
```

```
# Візуалізація матриці змішування
cm = confusion_matrix(y_test, y_test_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()

# Перевірка кількості зразків у кожному класі
class_counts = Counter(y)
min_class_samples = min(class_counts.values())

# Потрійна перехресна перевірка з використанням різних метрик
num_folds = min(3,
                min_class_samples)  # Забезпечуємо, щоб num_folds не перевищу-
вало кількість зразків у найменшому класі
if num_folds > 1:
    kfold = StratifiedKFold(n_splits=num_folds)

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

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

    recall_values = cross_val_score(classifier_new, X, y, scoring='recall_weighted',
cv=kfold, error_score='raise')
    print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")
```

```
    f1_values = cross_val_score(classifier_new, X, y, scoring='f1_weighted',
cv=kfold, error_score='raise')
    print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")
else:
    print("Недостатньо даних для виконання крос-валідації.")
```

Результат виконання роботи програми:

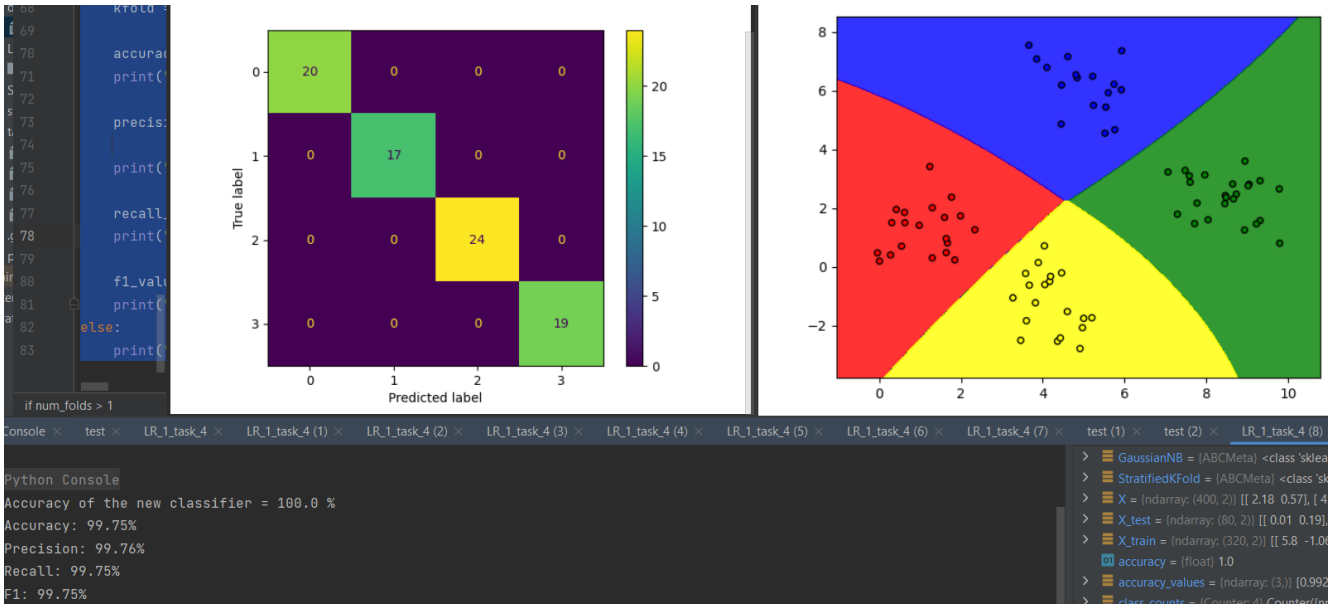
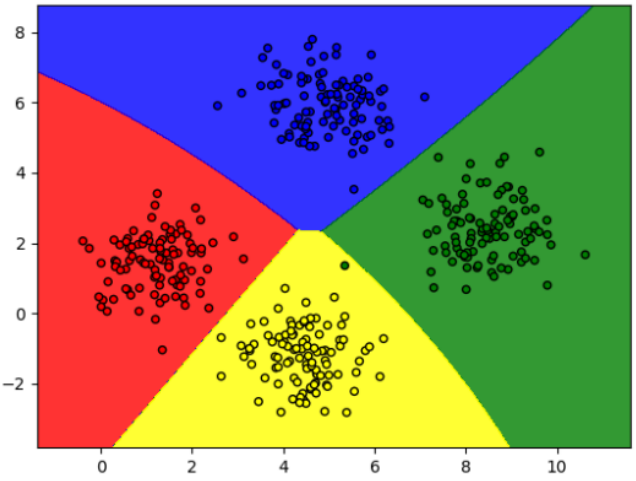
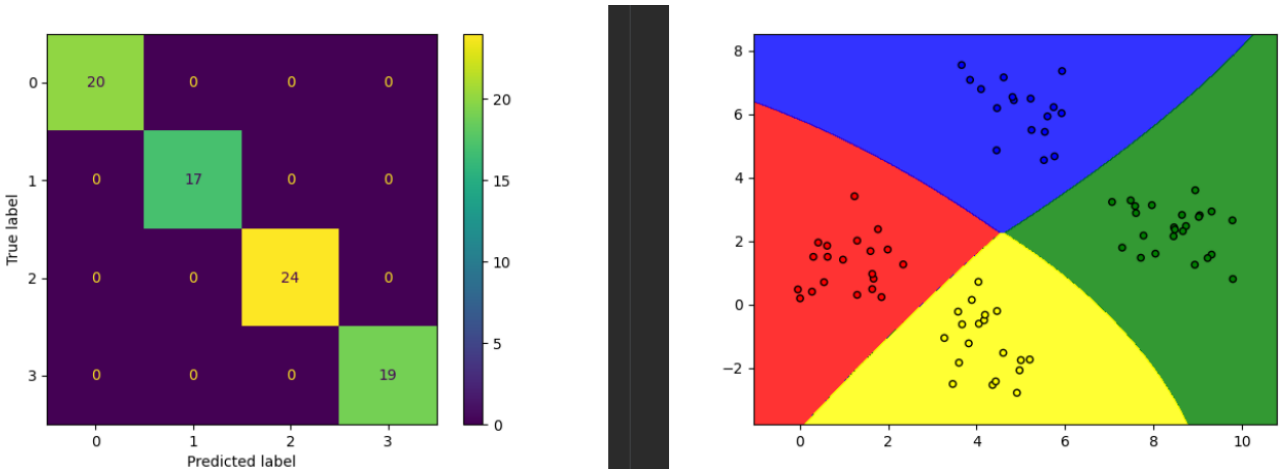


Рисунок 6 — Результат виконання програми.

Зробіть ще один прогін та зображення результатів класифікації занесіть у звіт. (рис. 7 звіту)



Accuracy of Naive Bayes classifier = 99.75 %



Accuracy of the new classifier = 100.0 %

Accuracy: 99.75%

Precision: 99.76%

Recall: 99.75%

F1: 99.75%

Рисунок 7 — Ще один прогін програми і порівняння результатів.

Порівняйте між собою результати висновок запишіть у звіт:

Другий код дає точніші результати, оскільки використовує унікальний тестовий зразок і перехресну перевірку, що дозволяє уникнути надмірного навчання. Точність моделі з оригінальним кодом становить 99,75%, але вона не використовує додаткових тестів, тому результат може бути значним. У другому коді точність на тестовій вибірці становить 100%, а після перехресної перевірки — 99,75%, що свідчить про надійність моделі. Крім того, показники точності, Precision, Recall та оцінки F1 вказують на ефективність класифікації для кожного класифікатора.

Завдання 2.5. Вивчити метрики якості класифікації.

Код програми:

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

# Завантаження даних
df = pd.read_csv('../data/data_metrics.csv')

# Додавання стовпців predicted_RF та predicted_LR на основі порогу 0.5
thresh = 0.5
df['predicted_RF'] = (df.model_RF >= thresh).astype('int')
df['predicted_LR'] = (df.model_LR >= thresh).astype('int')

# Функції для обчислення TP, FN, FP, TN
def kerest_find_TP(y_true, y_pred):
    return sum((y_true == 1) & (y_pred == 1))

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

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

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

# Функція для обчислення значень матриці сплутаних результатів
```

```
def kerest_find_conf_matrix_values(y_true, y_pred):
    TP = kerest_find_TP(y_true, y_pred)
    FN = kerest_find_FN(y_true, y_pred)
    FP = kerest_find_FP(y_true, y_pred)
    TN = kerest_find_TN(y_true, y_pred)
    return TP, FN, FP, TN

# Функція для створення матриці сплутаних результатів
def kerest_confusion_matrix(y_true, y_pred):
    TP, FN, FP, TN = kerest_find_conf_matrix_values(y_true, y_pred)
    return np.array([[TN, FP], [FN, TP]])

# Функція для обчислення точності
def kerest_accuracy_score(y_true, y_pred):
    TP, FN, FP, TN = kerest_find_conf_matrix_values(y_true, y_pred)
    return (TP + TN) / (TP + TN + FP + FN)

# Функція для обчислення recall
def kerest_recall_score(y_true, y_pred):
    TP, FN, FP, TN = kerest_find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FN)

# Функція для обчислення precision
def kerest_precision_score(y_true, y_pred):
    TP, FN, FP, TN = kerest_find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FP)

# Функція для обчислення F1-score
def kerest_f1_score(y_true, y_pred):
    recall = kerest_recall_score(y_true, y_pred)
```

```
    precision = kerest_precision_score(y_true, y_pred)
    return 2 * (precision * recall) / (precision + recall)

# Перевірка результатів для моделі LR
print('TP (LR):', kerest_find_TP(df.actual_label.values, df.predicted_LR.values))
print('FN (LR):', kerest_find_FN(df.actual_label.values, df.predicted_LR.values))
print('FP (LR):', kerest_find_FP(df.actual_label.values, df.predicted_LR.values))
print('TN (LR):', kerest_find_TN(df.actual_label.values, df.predicted_LR.values))

# Перевірка матриці сплутаних результатів та точності для LR
assert np.array_equal(kerest_confusion_matrix(df.actual_label.values,
df.predicted_LR.values),
                      confusion_matrix(df.actual_label.values, df.predicted_LR.values)),
'Confusion matrix LR incorrect'
assert kerest_accuracy_score(df.actual_label.values, df.predicted_LR.values) ==
accuracy_score(df.actual_label.values, df.predicted_LR.values), 'Accuracy LR
incorrect'
assert kerest_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values, df.predicted_LR.values), 'F1 LR incorrect'

# ROC-крива для обох моделей
fpr_RF, tpr_RF, _ = roc_curve(df.actual_label.values, df.model_RF.values)
fpr_LR, tpr_LR, _ = roc_curve(df.actual_label.values, df.model_LR.values)

# Побудова ROC-кривої для RF та LR
plt.plot(fpr_RF, tpr_RF, 'r-', label='RF AUC: %.3f' %
roc_auc_score(df.actual_label.values, df.model_RF.values))
plt.plot(fpr_LR, tpr_LR, 'b-', label='LR AUC: %.3f' %
roc_auc_score(df.actual_label.values, df.model_LR.values))
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.title('ROC Curves for RF and LR Models')
plt.show()
```

Результат виконання програми:

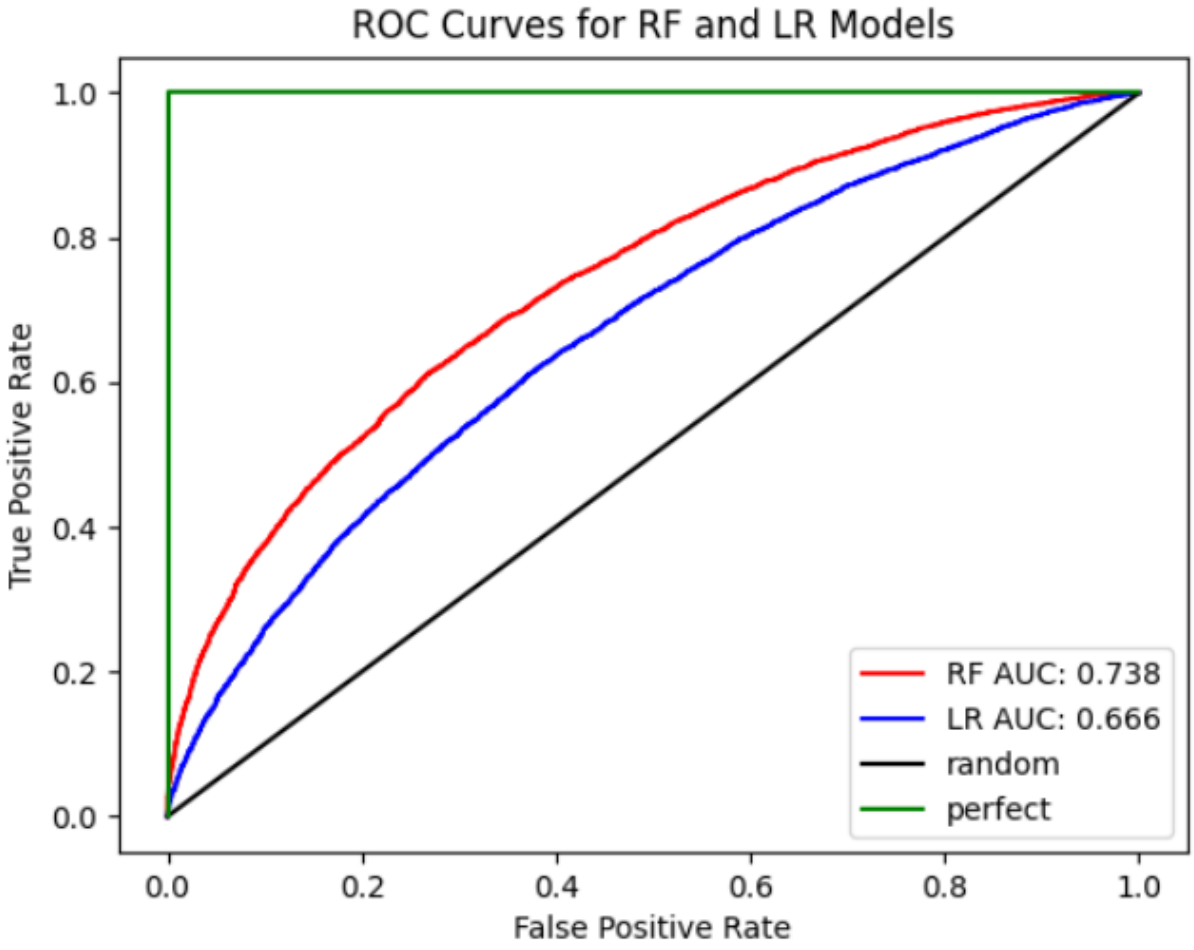


Рисунок 8 — Результат виконання програми.

Порівняйте результати для різних порогів та зробіть висновки.

Results for Random Forest (RF):

	Threshold	Accuracy	Precision	Recall	F1-Score
0	0.1	0.5	0.500000	1.0	0.666667
1	0.2	0.6	0.555556	1.0	0.714286
2	0.3	0.9	0.833333	1.0	0.909091
3	0.4	0.9	0.833333	1.0	0.909091
4	0.5	1.0	1.000000	1.0	1.000000
5	0.6	1.0	1.000000	1.0	1.000000
6	0.7	0.9	1.000000	0.8	0.888889
7	0.8	0.8	1.000000	0.6	0.750000
8	0.9	0.7	1.000000	0.4	0.571429

Results for Logistic Regression (LR):

	Threshold	Accuracy	Precision	Recall	F1-Score
0	0.1	0.6	0.555556	1.0	0.714286
1	0.2	0.7	0.625000	1.0	0.769231
2	0.3	0.9	0.833333	1.0	0.909091
3	0.4	0.9	0.833333	1.0	0.909091
4	0.5	1.0	1.000000	1.0	1.000000
5	0.6	1.0	1.000000	1.0	1.000000
6	0.7	0.8	1.000000	0.6	0.750000
7	0.8	0.7	1.000000	0.4	0.571429
8	0.9	0.6	1.000000	0.2	0.333333

Рисунок 9 — Порівняння результатів для різних порогів.

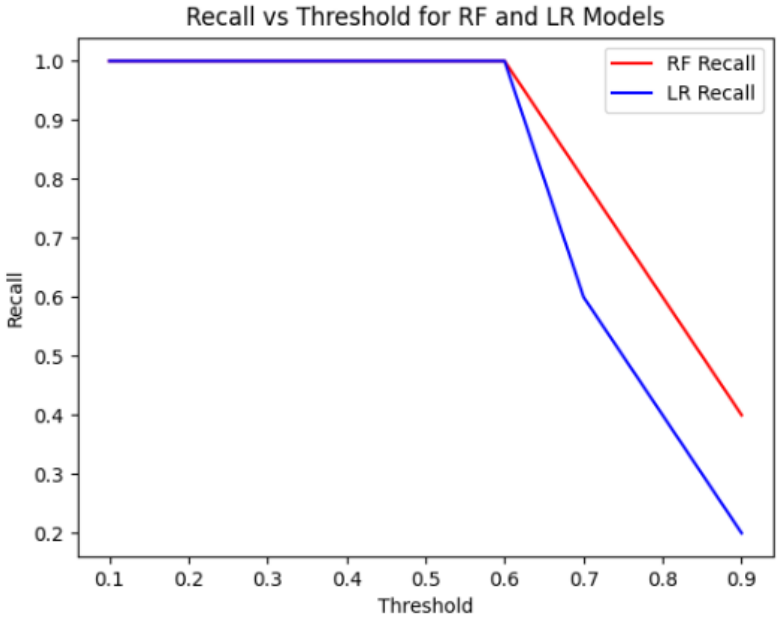


Рисунок 10 — Побудування кривої.

Висновки:

Аналіз результатів показує, що для двох моделей (Random Forest та Logistic Regression) при пороговому значенні 0,5 досягаються ідеальна точність, precision, recall і F1-оцінка, які відрізняються від оптимальної продуктивності класифікації. Однак зі збільшенням порогу точність моделей падає, кількість правильних випадків (recall) зменшується. Це особливо очевидно для значень вище 0,7, де Random Forest продовжує працювати краще через логістичну регресію, але обидві моделі показують зниження показника F1. Це свідчить про те, що поріг важливий для балансу між точністю та

запам’ятовуванням, і поріг 0,5 може бути найкращим для обох моделей у цій роботі.

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.

Програмний код:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Завантаження даних
data = pd.read_csv('../data/data_multivar_nb.txt', header=None)
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Розбиття на навчальну і тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Машина опорних векторів (SVM)
svm_model = SVC()
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)
```

```
# Наївний байєсівський класифікатор
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)

# Оцінка моделей
print("SVM Model Performance")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_svm))
print("Classification Report:")
print(classification_report(y_test, y_pred_svm))

print("\nNaive Bayes Model Performance")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_nb))
print("Classification Report:")
print(classification_report(y_test, y_pred_nb))

# Візуалізація матриці плутанини для SVM
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, fmt='d',
cmap='Blues')
plt.title('Confusion Matrix for SVM')

# Візуалізація матриці плутанини для Naive Bayes
plt.subplot(1, 2, 2)
sns.heatmap(confusion_matrix(y_test, y_pred_nb), annot=True, fmt='d',
```

cmap='Greens')

plt.title('Confusion Matrix for Naive Bayes')

plt.show()

Результат виконання програми:

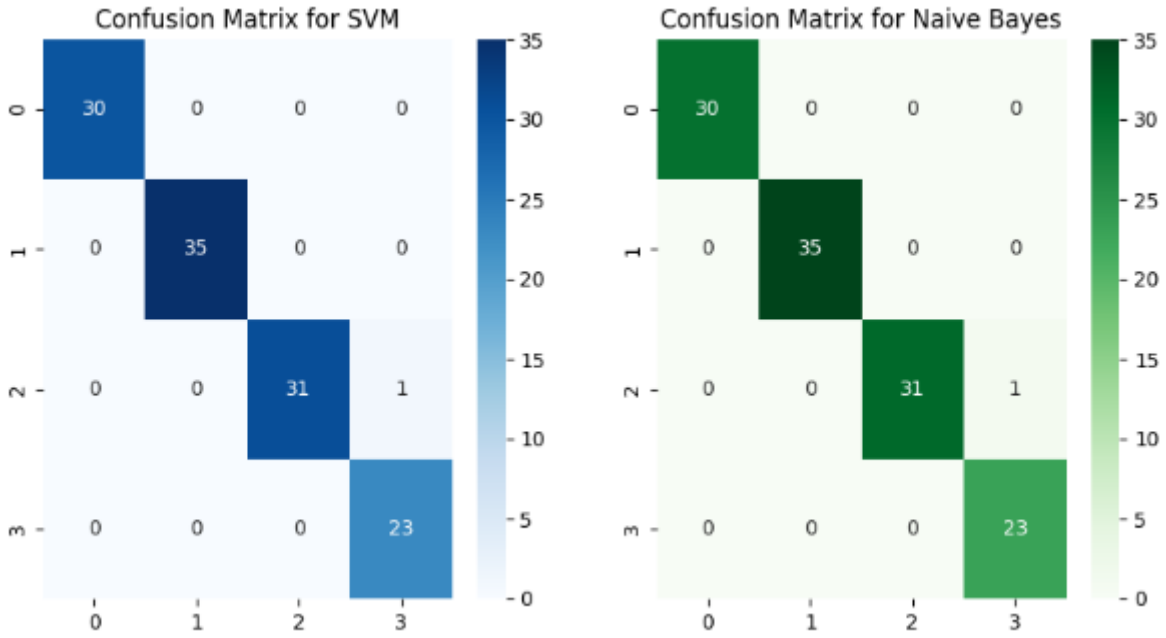


Рисунок 11 — Результат виконання програми(графічно).

Вивід консолі:

SVM Model Performance

Accuracy: 0.9916666666666667

Confusion Matrix:

[[30 0 0 0]

[0 35 0 0]

[0 0 31 1]

[0 0 0 23]]

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	30
1	1.00	1.00	1.00	35
2	1.00	0.97	0.98	32
3	0.96	1.00	0.98	23
accuracy			0.99	120
macro avg	0.99	0.99	0.99	120
weighted avg	0.99	0.99	0.99	120

Naive Bayes Model Performance

Accuracy: 0.9916666666666667

Confusion Matrix:

[[30 0 0 0]

[0 35 0 0]

[0 0 31 1]

[0 0 0 23]]

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	30
1	1.00	1.00	1.00	35
2	1.00	0.97	0.98	32
3	0.96	1.00	0.98	23
accuracy			0.99	120

macro avg	0.99	0.99	0.99	120
weighted avg	0.99	0.99	0.99	120

Зробіть висновки яку модель класифікації краще обрати і чому.

Обидві моделі — машина опорних векторів (SVM) і наївний байєсівський класифікатор — показали майже однакові результати, з точністю 99,17%. Матриці плутанини та ключові показники, такі як precision, recall і f1-score, також практично не відрізняються. Однак SVM краще підходить для складних задач, оскільки вона може побудувати складніші межі між класами і є більш універсальною моделлю. Наївний байєсівський класифікатор простіший у використанні та швидший, але робить припущення про незалежність ознак, що може бути проблемою для складніших даних. Тому, якщо важлива точність на складних даних, варто обрати SVM, а якщо потрібна швидкість і простота — наївний байєс.

Висновок: Під час виконання лабораторної роботи я використовував спеціалізовані бібліотеки та мову програмування Python, дослідив попередню обробку та класифікацію даних.

Посилання на GitHub: *натисніть*