ЛАБОРАТОРНА РОБОТА № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

Хід роботи:

Завдання №1: Попередня обробка даних.

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: ")
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)
```

```
 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]]

Process finished with exit code 0
```

Рис. 1. Результати виконання програми (Попередня обробка даних)

L1-нормалізація використовує метод найменших абсолютних відхилень, що забезпечує рівність 1 суми абсолютних значень в кожному ряду, в той час як, **L2-нормалізація –** рівність 1 суми квадратів значень в кожному ряду. Тобто, для

L1-нормалізації в першому рядку буде |0.45| + |-0.25| + |0.3| = 1 (значення заокруглені для наочності). А для **L2-нормалізації** – $0.75^2 + (-0.43)^2 + (0.49)^2 \approx 1$ (значення заокруглені для наочності).

Лістинг програми:

```
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 чис-
лами
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))
```

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

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

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

Process finished with exit code 0
```

Рис. 2. Результати виконання програми (Кодування міток)

Завдання №2: Попередня обробка нових даних.

Таблиця 1

№ варіанту	Значення змінної input_data												Поріг бінаризації
3	-1.3	3.9	4.5	-5.3	-4.2	3.3	-5.2	-6.5	-1.1	-5.2	2.6	-2.2	1.8

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([
    [-1.3, 3.9, 4.5],
    [-5.3, -4.2, 3.3],
    [-5.2, -6.5, -1.1],
    [-5.2, 2.6, -2.2]
])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=1.8).transform(in-
put 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)
```

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

BEFORE:
Mean = [-4.25  -1.05   1.125]
Std deviation = [1.7036725  4.40028408 2.83405628]

AFTER:
Mean = [0.00000000e+00 5.55111512e-17 0.00000000e+00]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[1.         1.         1.        ]
 [0.         0.22115385 0.82089552]
 [0.025      0.         0.1641791 ]
 [0.025      0.875      0.        ]]

l1 normalized data:
 [[-0.13402062  0.40206186  0.46391753]
 [-0.4140625  -0.328125    0.2578125 ]
 [-0.40625    -0.5078125  -0.0859375 ]
 [-0.52        0.26       -0.22      ]]

l2 normalized data:
 [[-0.21328678  0.63986035  0.7383004 ]
 [-0.70435392 -0.55816726  0.43855999]
 [-0.61931099 -0.77413873 -0.13100809]
 [-0.83653629  0.41826814 -0.3539192 ]]

Process finished with exit code 0
```

Рис. 3. Результати виконання програми (Попередня обробка даних)

Завдання №3: Класифікація логістичною регресією або логістичний класифікатор.

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)
```

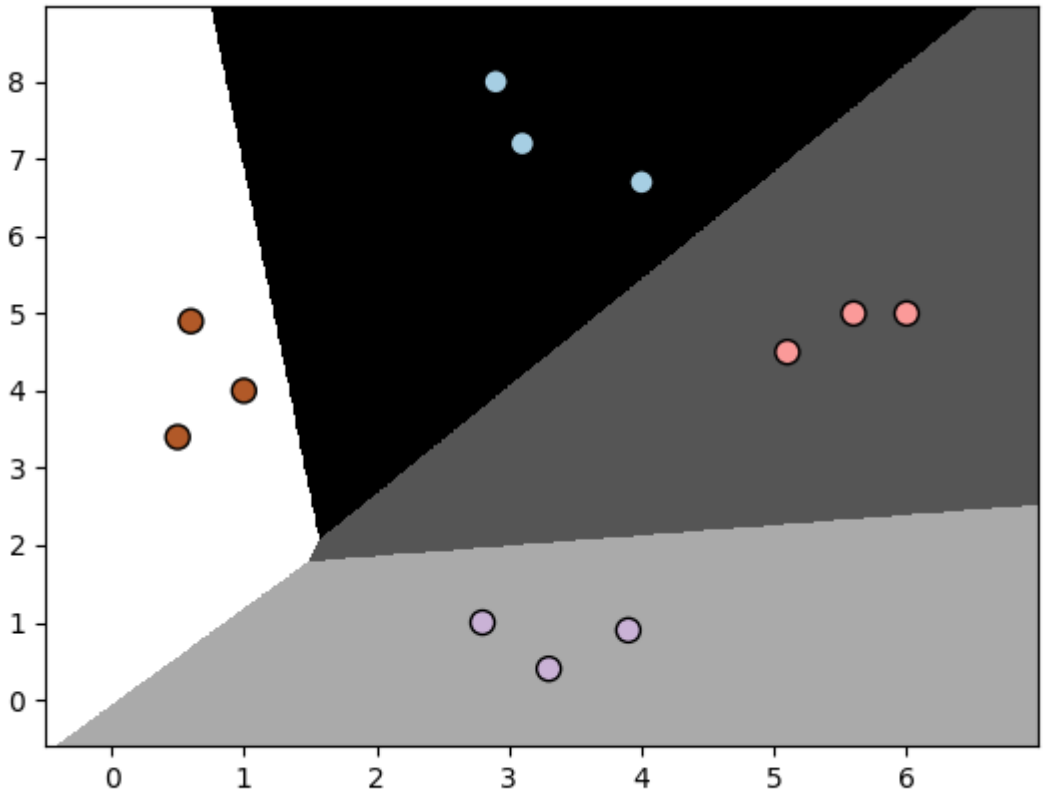
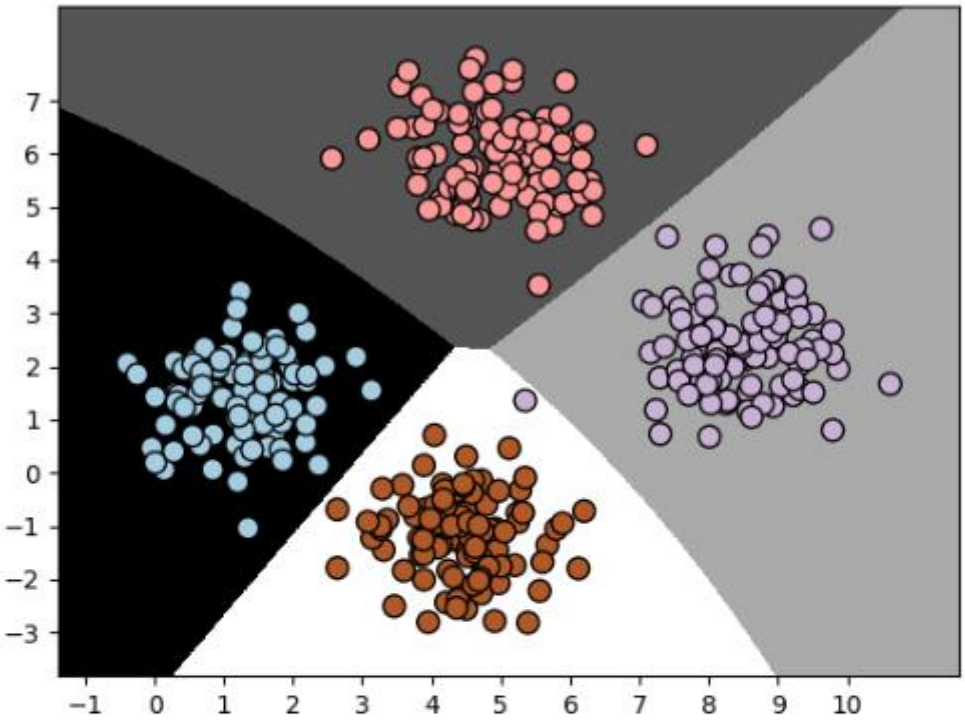


Рис. 4. Результати виконання програми (Логістичний класифікатор)

Завдання №4: Класифікація наївним байєсовським класифікатором.

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)
```



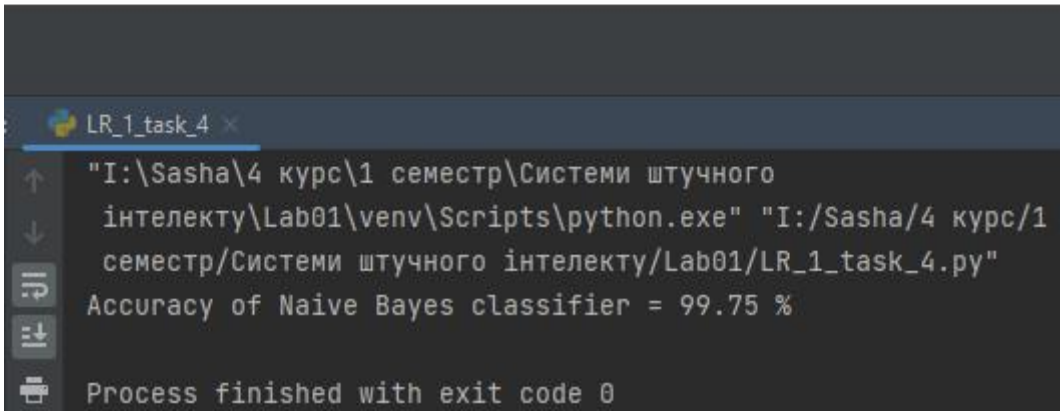


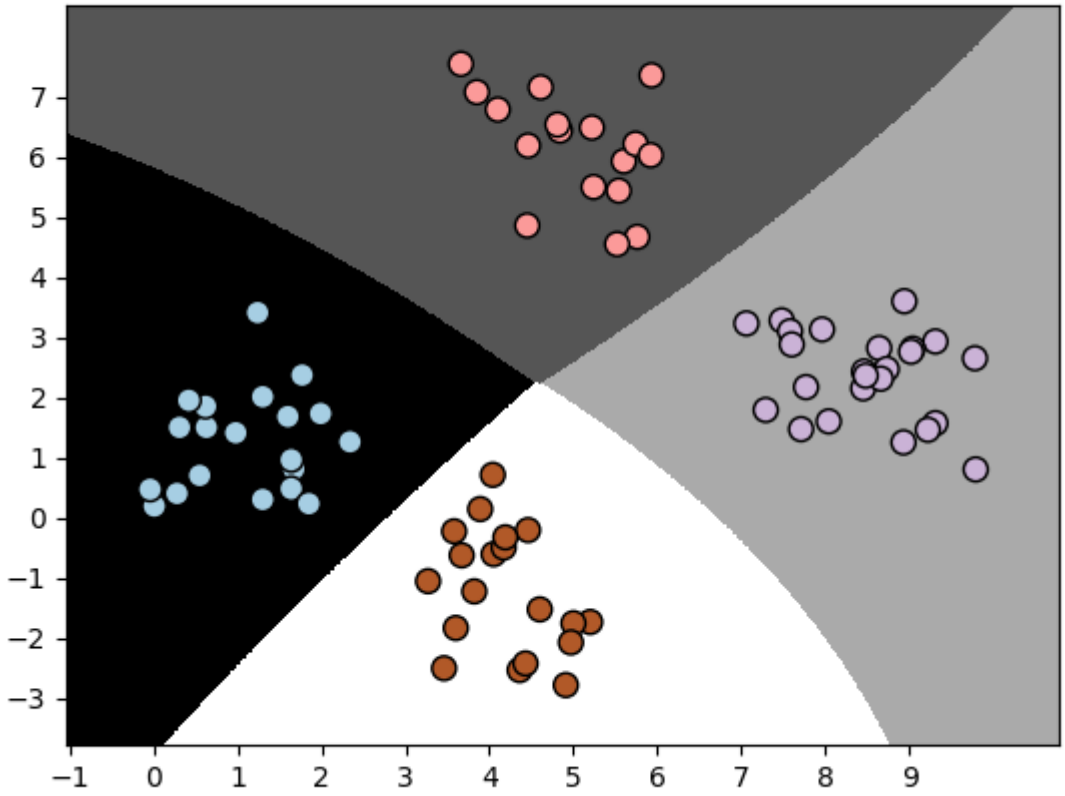
Рис. 5. Результати виконання програми

(Класифікація наївним байєсовським класифікатором)

Лістинг програми:

```
# 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)
classifier_new = GaussianNB()
classifier_new.fit(X_train, y_train)
y_test_pred = classifier_new.predict(X_test)

# Обчислення якості класифікатора
accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print("Accuracy of the new classifier =", round(accuracy, 2), "%")

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

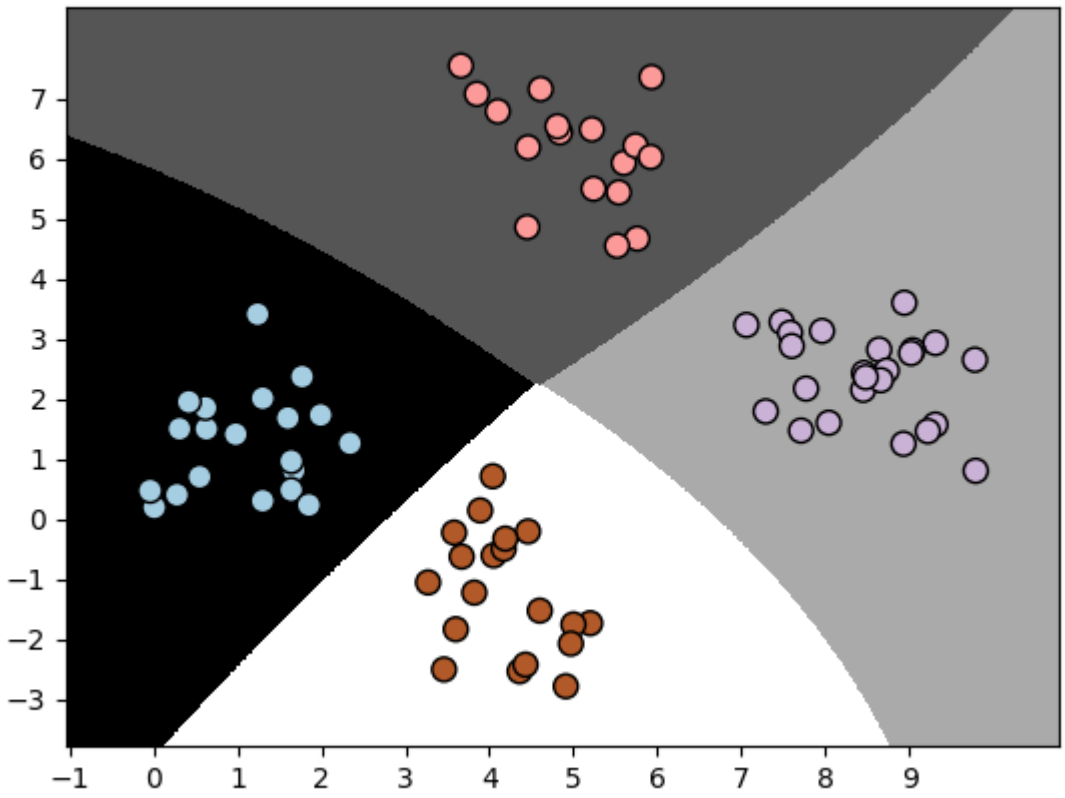
```
num_folds = 3
accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy',
cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
precision_values = cross_val_score(classifier, X, y, scoring='preci-
sion_weighted', cv=num_folds)
print("Precision: " + str(round(100 * 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)) + "%")
```




```
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%
```

Рис. 6. Результати виконання програми

(1 прогін)




```
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%
```

Рис. 7. Результати виконання програми

(2 прогін)

Отримані результати після двох прогонів ідентичні, бо тренування відбувалися на однакових початкових значеннях.

Завдання №5: Вивчити метрики якості класифікації.

Лістинг програми:

```
import pandas as pd
from sklearn.metrics import confusion_matrix
import numpy as np
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

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


def find_TP(y_true, y_pred):
    # counts the number of true positives (y_true = 1, y_pred = 1)
    return sum((y_true == 1) & (y_pred == 1))


def find_FN(y_true, y_pred):
    # counts the number of false negatives (y_true = 1, y_pred = 0)
    return sum((y_true == 1) & (y_pred == 0))


def find_FP(y_true, y_pred):
    # counts the number of false positives (y_true = 0, y_pred = 1)
    return sum((y_true == 0) & (y_pred == 1))


def find_TN(y_true, y_pred):
    # counts the number of true negatives (y_true = 0, y_pred = 0)
    return sum((y_true == 0) & (y_pred == 0))


# print('TP:', find_TP(df.actual_label.values, df.predicted_RF.values))
# print('FN:', find_FN(df.actual_label.values, df.predicted_RF.values))
# print('FP:', find_FP(df.actual_label.values, df.predicted_RF.values))
# print('TN:', find_TN(df.actual_label.values, df.predicted_RF.values))


def find_conf_matrix_values(y_true, y_pred):
    # calculate TP, FN, FP, TN
    TP = find_TP(y_true, y_pred)
    FN = find_FN(y_true, y_pred)
    FP = find_FP(y_true, y_pred)
    TN = find_TN(y_true, y_pred)
    return TP, FN, FP, TN


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


print(svistelnyk_confusion_matrix(df.actual_label.values, df.pre-
dicted_RF.values))
print(svistelnyk_confusion_matrix(df.actual_label.values, df.pre-
dicted_LR.values))
assert np.array_equal(svistelnyk_confusion_matrix(df.actual_label.values,
df.predicted_RF.values),
```

```
                      confusion_matrix(df.actual_label.values,
                                       df.predicted_RF.values)), 'svistel-
nyk_confusion_matrix() is not correct for RF'
assert np.array_equal(svistelnyk_confusion_matrix(df.actual_label.values,
df.predicted_LR.values),
                      confusion_matrix(df.actual_label.values,
                                       df.predicted_LR.values)), 'svistel-
nyk_confusion_matrix() is not correct for LR'

print(accuracy_score(df.actual_label.values, df.predicted_RF.values))
print(accuracy_score(df.actual_label.values, df.predicted_LR.values))


def svistelnyk_accuracy_score(y_true, y_pred):
    # calculates the fraction of samples
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return (TP + TN) / (TP + TN + FP + FN)


assert svistelnyk_accuracy_score(df.actual_label.values, df.pre-
dicted_RF.values) == accuracy_score(
    df.actual_label.values,
    df.predicted_RF.values), 'svistelnyk_accuracy_score failed on assert
svistelnyk_accuracy_score(df.actual_label.values, ' \
                             'df.predicted_LR.values) == accu-
racy_score(df.actual_label.values, df.predicted_LR.values),' \
                             'svistelnyk_accuracy_score failed on LR'
print('Accuracy RF: % .3f' % (svistelnyk_accuracy_score(df.actual_label.val-
ues, df.predicted_RF.values)))
print('Accuracy LR: % .3f' % (svistelnyk_accuracy_score(df.actual_label.val-
ues, df.predicted_LR.values)))

print(recall_score(df.actual_label.values, df.predicted_RF.values))
print(recall_score(df.actual_label.values, df.predicted_LR.values))


def svistelnyk_recall_score(y_true, y_pred):
    # calculates the fraction of positive samples predicted correctly
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FN)


assert svistelnyk_recall_score(df.actual_label.values, df.predicted_RF.val-
ues) == recall_score(df.actual_label.values,

df.predicted_RF.values), \
    'svistelnyk_recall_score failed on RF'
assert svistelnyk_recall_score(df.actual_label.values, df.predicted_LR.val-
ues) == recall_score(df.actual_label.values,

df.predicted_LR.values), \
    'svistelnyk_recall_score failed on LR'
print('Recall RF: %.3f' % (svistelnyk_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall LR: %.3f' % (svistelnyk_recall_score(df.actual_label.values,
df.predicted_LR.values)))

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


def svistelnyk_precision_score(y_true, y_pred):
```

```
    # calculates the fraction of predicted positives samples that are actu-
ally positive
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FP)


assert svistelnyk_precision_score(df.actual_label.values, df.pre-
dicted_RF.values) == precision_score(
    df.actual_label.values, df.predicted_RF.values), 'svistelnyk_preci-
sion_score failed on RF'
assert svistelnyk_precision_score(df.actual_label.values, df.pre-
dicted_LR.values) == precision_score(
    df.actual_label.values, df.predicted_LR.values), 'svistelnyk_preci-
sion_score failed on LR'
print('Precision RF: %.3f' % (svistelnyk_precision_score(df.actual_la-
bel.values, df.predicted_RF.values)))
print('Precision LR: %.3f' % (svistelnyk_precision_score(df.actual_la-
bel.values, df.predicted_LR.values)))

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


def svistelnyk_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = svistelnyk_recall_score(y_true, y_pred)
    precision = svistelnyk_precision_score(y_true, y_pred)
    return (2 * (precision * recall)) / (precision + recall)


assert svistelnyk_f1_score(df.actual_label.values, df.predicted_RF.values)
== f1_score(df.actual_label.values,

df.predicted_RF.values), 'svistelnyk_f1_score failed on RF'
assert svistelnyk_f1_score(df.actual_label.values, df.predicted_LR.values)
== f1_score(df.actual_label.values,

df.predicted_LR.values), 'svistelnyk_f1_score failed on LR'
print('F1 RF: %.3f' % (svistelnyk_f1_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('F1 LR: %.3f' % (svistelnyk_f1_score(df.actual_label.values, df.pre-
dicted_LR.values)))

print('scores with threshold = 0.5')
print('Accuracy RF: %.3f' % (svistelnyk_accuracy_score(df.actual_label.val-
ues, df.predicted_RF.values)))
print('Recall RF: %.3f' % (svistelnyk_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: %.3f' % (svistelnyk_precision_score(df.actual_la-
bel.values, df.predicted_RF.values)))
print('F1 RF: %.3f' % (svistelnyk_f1_score(df.actual_label.values, df.pre-
dicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f' % (
    svistelnyk_accuracy_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('Recall RF: %.3f' % (svistelnyk_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f' % (
    svistelnyk_precision_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
```

```
print('F1 RF: %.3f' % (svistelnyk_f1_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
```

```
[[5519 2360]
 [2832 5047]]
[[5425 2454]
 [3600 4279]]
[[5519 2360]
 [2832 5047]]
[[5425 2454]
 [3600 4279]]
0.6705165630156111
0.6158141896179719
Accuracy RF:  0.671
Accuracy LR:  0.616
0.6405635232897576
0.5430892245208783
Recall RF: 0.641
Recall LR: 0.543
0.681382476036182
0.6355265112134264
Precision RF: 0.681
Precision LR: 0.636
0.660342797330891
0.5856830002737475
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

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

Рис. 8. Результати виконання програми

(Метрики якості класифікації)

Акуратність для більшого порогу є кращою за акуратність для меншого. Але чутливість навпаки для більшого порогу є меншою. Точність для порогу 0.5 виявилася більшою за точність для 0.25. Оцінка f1 є майже ідентичною для обох порогів, а так як цей показник є одним з точніших для визначення пріоритетної моделі, то можемо зробити висновок, що обидва пороги мають право життя.

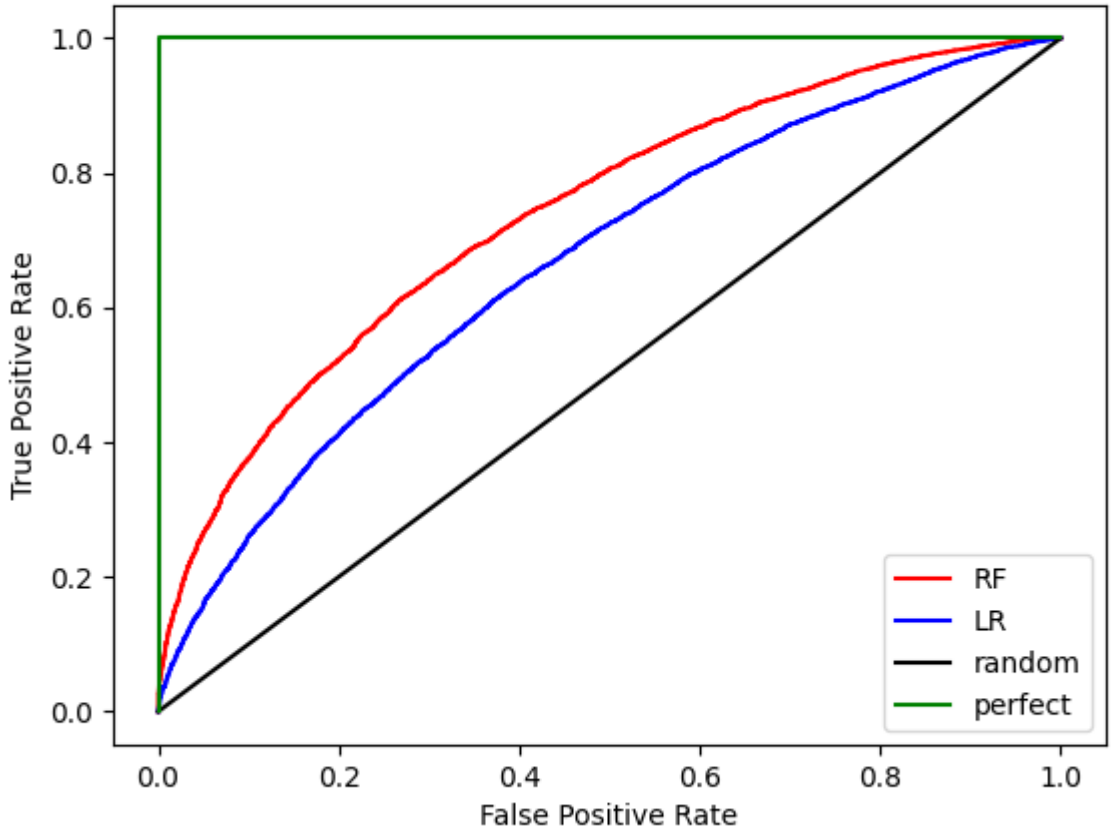


Рис. 9. Крива ROC для обох моделей

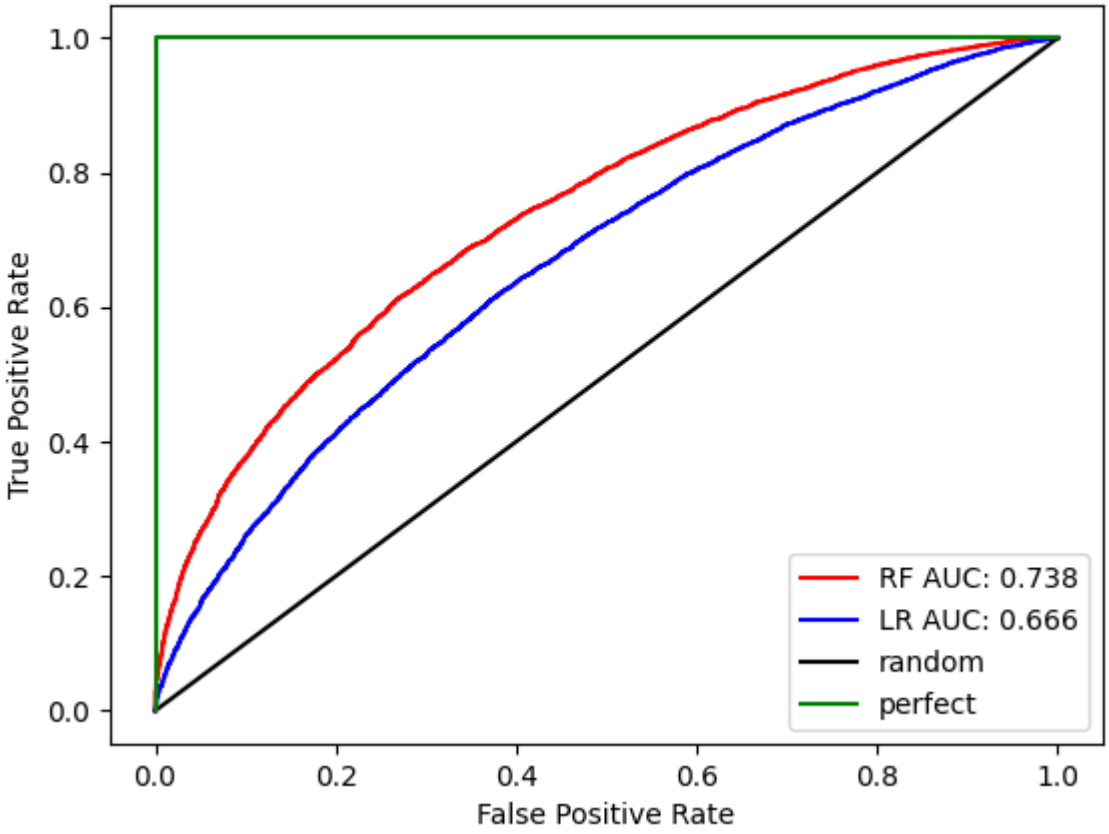


Рис. 10. Крива ROC для обох моделей (з урахуванням площ під кривою)

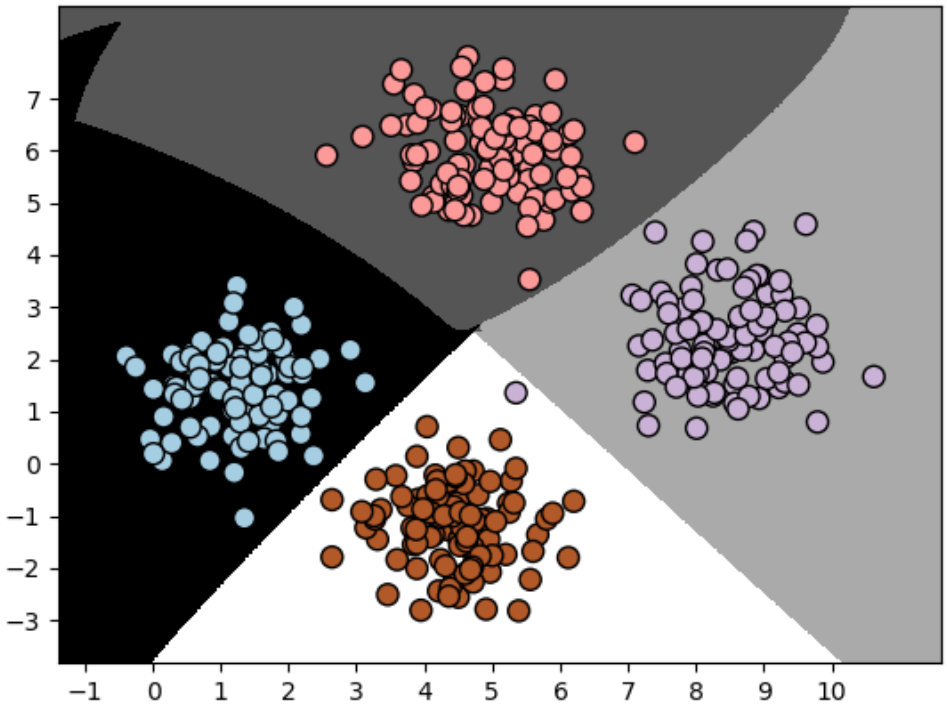
Площа під кривою для моделі RF (AUC = 0,738) краще, ніж LR (AUC = 0,666). Отже, згідно вищевказаної метрики робимо висновок, що модель RF краще.

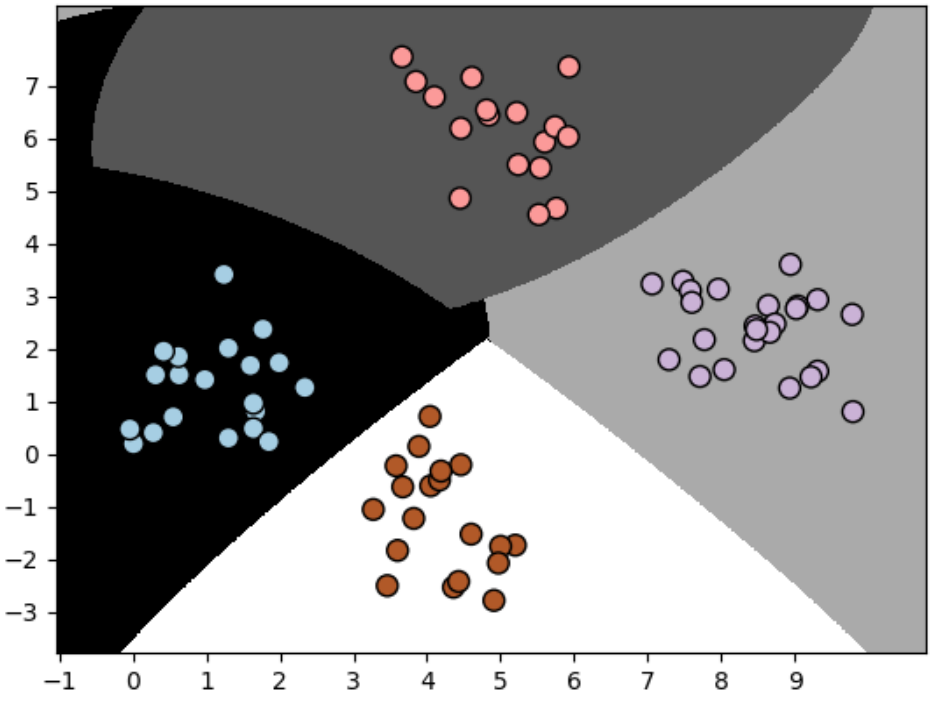
Завдання №6: Розробіть програму класифікації даних в файлі data_multivar_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVM). Розрахуйте показники якості класифікації. Порівняйте їх з показниками наївного байєсівського класифікатора. Зробіть висновки яку модель класифікації краще обрати і чому.

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.model_selection import 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 класифікатора SVM
classifier = SVC()

# 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 Support Vector Machine classifier =", round(accuracy, 2),
"%")

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

# 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)
classifier_new = SVC()
classifier_new.fit(X_train, y_train)
y_test_pred = classifier_new.predict(X_test)

# Обчислення якості класифікатора
accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print("Accuracy of the new classifier =", round(accuracy, 2), "%")

# Візуалізація роботи класифікатора
visualize_classifier(classifier_new, X_test, y_test)

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='preci-
sion_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)) + "%")
```






```
Accuracy of Support Vector Machine classifier = 99.75 %
Accuracy of the new classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Рис. 11. Результати виконання програми

(Класифікація Support Vector Machine - SVM)

Показники отримані з показниками обох класифікаторів ідентичні. Тому визначити який класифікатор краще неможливо на даному прикладі. Але зважаючи на те, що наївний байєсівський класифікатор визначає кожну ознаку як незалежну, важко отримати повну картину. Через це доцільніше використовувати класифікатор методу опорних векторів, а, також, він є найпопулярнішим методом класичної класифікації.

Посилання на GitHub: https://github.com/KniazhytsynaOlga/ArtificialIntelligenceSystems

Висновки: було досліджено попередню обробку та класифікацію даних, використовуючи спеціалізовані бібліотеки та мову програмування Python. Також, було створено власні функції для знаходження показників. Було досліджено необхідні бібліотеки для оптимальної обробки даних. Також, було використано мову програмування Python та бібліотеку matplotlib для графічного відображення отриманих даних.