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.

```
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]])
```

[1] ✓ 1.3s

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

[2] ✓ 0.0s

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

Рис. 1.1 – Результат виконання програми

```
    # 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))
```

[3] ✓ 0.0s

```
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.]
```

```
    # 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)
```

[4] ✓ 0.0s

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

Рис. 1.2 – Результат виконання програми

```
    # 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)
```

[5] ✓ 0.0s

```
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]]
```

Рис. 1.3 – Результат виконання програми

Отже, L1-нормалізація відрізняється від L2-нормалізації тим, що в першому сума абсолютних значень в ряді дорівнює 1, а в другому сума квадратів значень в ряді дорівнює 1.

Завдання 2: Кодування міток.

```
import numpy as np
from sklearn import preprocessing

# Надання позначок вхідних даних
input_labels = ["red", "black", "red", "green", "black", "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)
```

✓ 1.4s

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

Рис. 2.1 – Результат виконання програми

```
# перетворення міток за допомогою кодувальника
test_labels = ["green", "red", "black"]
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))
```

[3] ✓ 0.0s

```
Labels = ['green', 'red', 'black']
Encoded values = [1, 2, 0]

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

Рис. 2.2 – Результат виконання програми

Завдання 3: Попередня обробка нових даних.

```
import numpy as np
from sklearn import preprocessing

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]]
)
```

[1] ✓ 1.3s

```
# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.0).transform(input_data)
print("\n Binarized data:\n", data_binarized)
```

[2] ✓ 0.0s

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

Рис. 3.1 – Результат виконання програми

```
    # 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))
[3] ✓ 0.0s
...
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.]


    # 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)
[4] ✓ 0.0s
...
Min max scaled data:
 [[0.64356436 0.73737374 1.        ]
 [1.         0.56565657 0.07894737]
 [0.25742574 1.         0.81578947]
 [0.         0.         0.        ]]
```

Рис. 3.2 – Результат виконання програми

```
    # 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)
[5] ✓ 0.0s
...
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]]
```

Рис. 3.3 – Результат виконання програми

Завдання 4: Класифікація логістичною регресією або логістичний класифікатор.

```python
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)
```

[1] ✓ 3.6s

Рис. 4.1 – Результат виконання програми

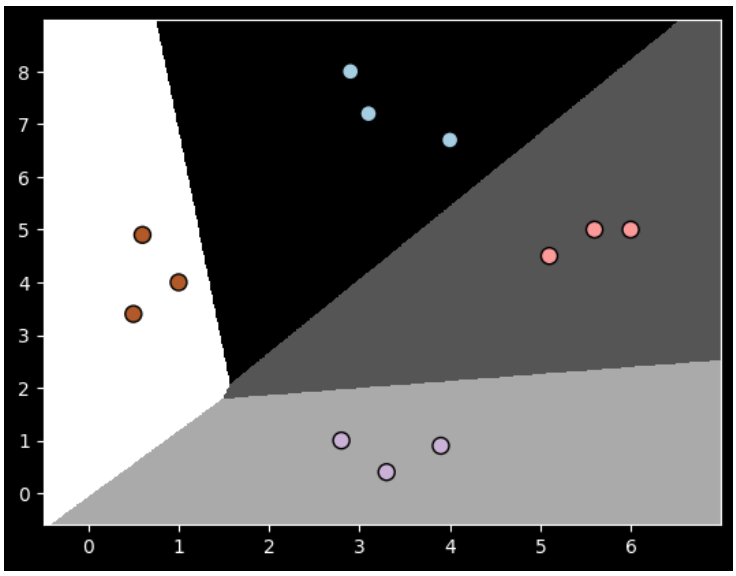

Рис. 4.2 – Результат виконання програми

Завдання 5: Класифікація наївним байєсовським класифікатором.

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

[1] ✓ 10.0s

Рис. 5.1 – Результат виконання програми

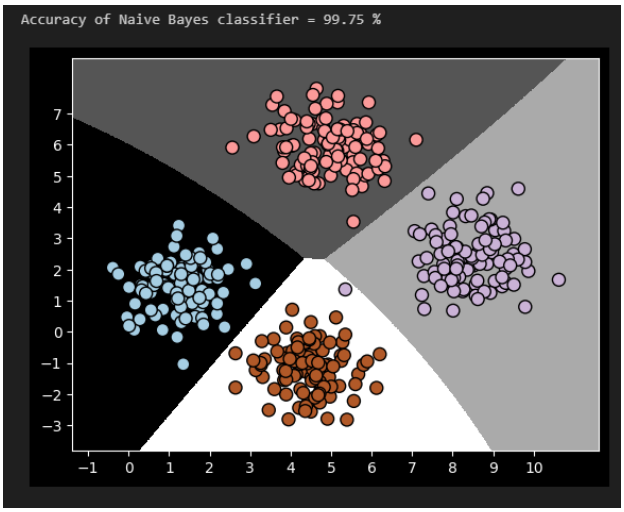

Рис. 5.2 – Результат виконання програми

```python
# 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="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)) + "%")
```

[2] ✓ 1.8s

Рис. 5.3 – Результат виконання програми

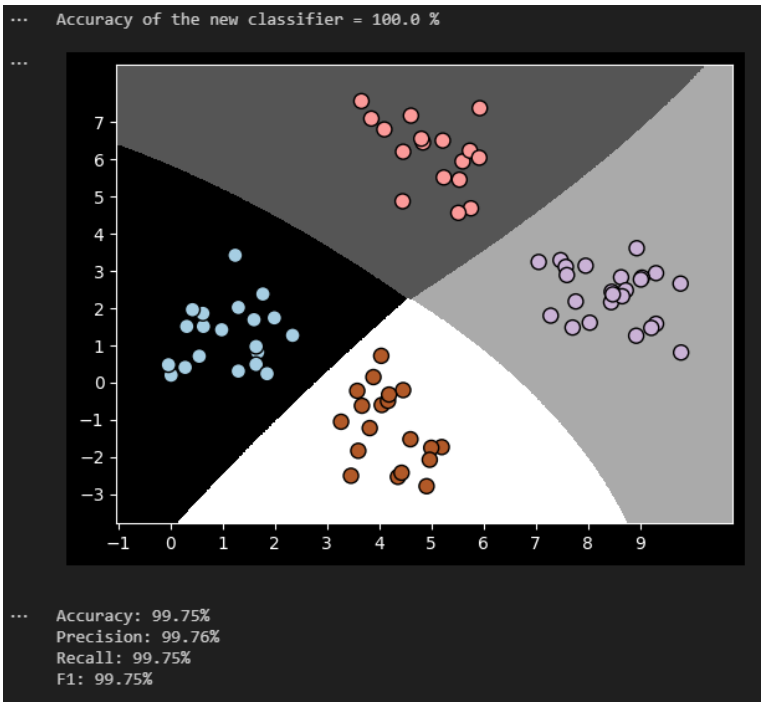

Рис. 5.4 – Результат виконання програми

Завдання 6: Вивчити метрики якості класифікації.

```
import pandas as pd

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()
```

[1] ✓ 4.4s

Launch Data Wrangler

	actual_label	model_RF	model_LR	predicted_RF	predicted_LR
0	1	0.639816	0.531904	1	1
1	0	0.490993	0.414496	0	0
2	1	0.623815	0.569883	1	1
3	1	0.506616	0.443674	1	0
4	0	0.418302	0.369532	0	0

Рис. 6.1 – Результат виконання програми

```
from sklearn.metrics import confusion_matrix

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

[2] ✓ 0.7s

```
array([[5519, 2360],
       [2832, 5047]], dtype=int64)
```

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


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


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


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

[3] ✓ 0.0s

```
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))
```

[4] ✓ 0.0s

```
TP: 5047
FN: 2832
FP: 2360
TN: 5519
```

Рис. 6.2 – Результат виконання програми

```
import numpy as np


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 skakovskyi_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]])
```

[5] ✓ 0.0s

```
assert np.array_equal(
    skakovskyi_confusion_matrix(df.actual_label.values, df.predicted_RF.values),
    confusion_matrix(df.actual_label.values, df.predicted_RF.values),
), "my_confusion_matrix() is not correct for RF"

assert np.array_equal(
    skakovskyi_confusion_matrix(df.actual_label.values, df.predicted_LR.values),
    confusion_matrix(df.actual_label.values, df.predicted_LR.values),
), "my_confusion_matrix() is not correct for LR"
```

[6] ✓ 0.0s

```
from sklearn.metrics import accuracy_score


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

[7] ✓ 0.0s

```
0.6705165630156111
```

Рис. 6.3 – Результат виконання програми

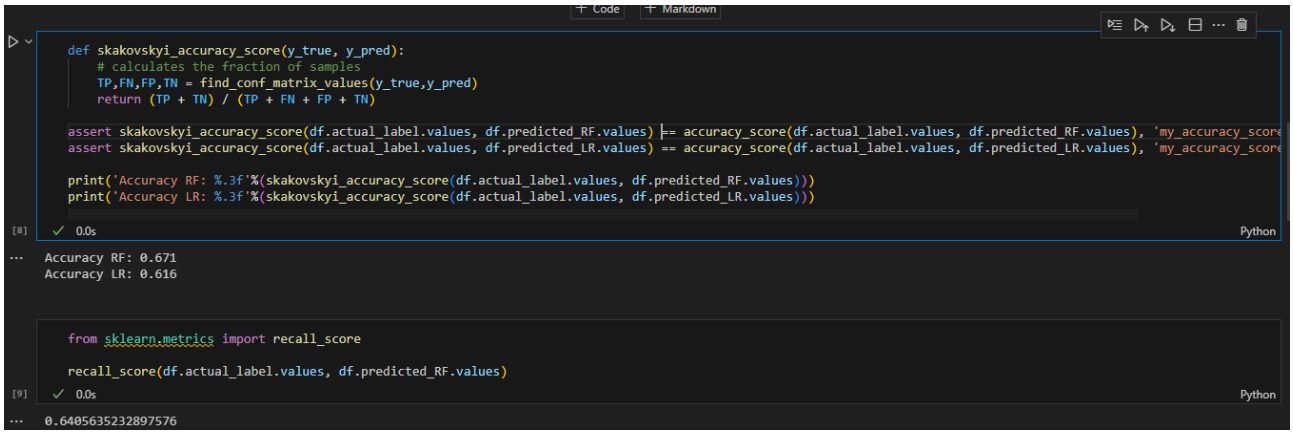

Рис. 6.4 – Результат виконання програми

```python
def skakovskyi_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 skakovskyi_recall_score(
    df.actual_label.values, df.predicted_RF.values
) == recall_score(
    df.actual_label.values, df.predicted_RF.values
), "my_accuracy_score failed on RF"
assert skakovskyi_recall_score(
    df.actual_label.values, df.predicted_LR.values
) == recall_score(
    df.actual_label.values, df.predicted_LR.values
), "my_accuracy_score failed on LR"

print(
    "Recall RF: %.3f"
    % (skakovskyi_recall_score(df.actual_label.values, df.predicted_RF.values))
)
print(
    "Recall LR: %.3f"
    % (skakovskyi_recall_score(df.actual_label.values, df.predicted_LR.values))
)
```

[10] ✓ 0.0s

```
Recall RF: 0.641
Recall LR: 0.543
```

```python
from sklearn.metrics import precision_score


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

[11] ✓ 0.0s

```
0.681382476036182
```

Рис. 6.5 – Результат виконання програми

```python
def skakovskyi_precision_score(y_true, y_pred):
    # calculates the fraction of predicted positives samples that are actually positive
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FP)


assert skakovskyi_precision_score(
    df.actual_label.values, df.predicted_RF.values
) == precision_score(
    df.actual_label.values, df.predicted_RF.values
), "my_accuracy_score failed on RF"


assert skakovskyi_precision_score(
    df.actual_label.values, df.predicted_LR.values
) == precision_score(
    df.actual_label.values, df.predicted_LR.values
), "my_accuracy_score failed on LR"

print(
    "Precision RF: %.3f"
    % (skakovskyi_precision_score(df.actual_label.values, df.predicted_RF.values))
)
print(
    "Precision LR: %.3f"
    % (skakovskyi_precision_score(df.actual_label.values, df.predicted_LR.values))
)
```

[12] ✓ 0.0s

```
Precision RF: 0.681
Precision LR: 0.636
```

```python
from sklearn.metrics import f1_score


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

[13] ✓ 0.0s

```
0.660342797330891
```

Рис. 6.6 – Результат виконання програми

```
def skakovskyi_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = skakovskyi_recall_score(y_true, y_pred)
    precision = skakovskyi_precision_score(y_true, y_pred)
    return 2 * (precision * recall) / (precision + recall)


assert skakovskyi_f1_score(df.actual_label.values, df.predicted_RF.values) == f1_score(
    df.actual_label.values, df.predicted_RF.values
), "my_accuracy_score failed on RF"
assert skakovskyi_f1_score(df.actual_label.values, df.predicted_LR.values) == f1_score(
    df.actual_label.values, df.predicted_LR.values
), "my_accuracy_score failed on LR"

print(
    "F1 RF: %.3f"
    % (skakovskyi_f1_score(df.actual_label.values, df.predicted_RF.values))
)
print(
    "F1 LR: %.3f"
    % (skakovskyi_f1_score(df.actual_label.values, df.predicted_LR.values))
)
```

[14] ✓ 0.1s

```
F1 RF: 0.660
F1 LR: 0.586
```

Рис. 6.7 – Результат виконання програми

```
print("scores with threshold = 0.5")

print(
    "Accuracy RF: %.3f"
    % (skakovskyi_accuracy_score(df.actual_label.values, df.predicted_RF.values))
)
print(
    "Recall RF: %.3f"
    % (skakovskyi_recall_score(df.actual_label.values, df.predicted_RF.values))
)
print(
    "Precision RF: %.3f"
    % (skakovskyi_precision_score(df.actual_label.values, df.predicted_RF.values))
)
print("F1 RF: %.3f" % (skakovskyi_f1_score(df.actual_label.values, df.predicted_RF.values)))
print("")
print("scores with threshold = 0.25")
print(
    "Accuracy RF: %.3f"
    % (
        skakovskyi_accuracy_score(
            df.actual_label.values, (df.model_RF >= 0.25).astype("int").values
        )
    )
)
print(
    "Recall RF: %.3f"
    % (
        skakovskyi_recall_score(
            df.actual_label.values, (df.model_RF >= 0.25).astype("int").values
        )
    )
)
print(
    "Precision RF: %.3f"
    % (
        skakovskyi_precision_score(
            df.actual_label.values, (df.model_RF >= 0.25).astype("int").values
        )
    )
)
print(
    "F1 RF: %.3f"
    % (skakovskyi_f1_score(df.actual_label.values, (df.model_RF >= 0.25).astype("int").values))
)
```

[15] ✓ 0.1s

```
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
```

Рис. 6.8 – Результат виконання програми

```
from sklearn.metrics import roc_curve

fpr_RF, tpr_RF, thresholds_RF = roc_curve(df.actual_label.values, df.model_RF.values)
fpr_LR, tpr_LR, thresholds_LR = roc_curve(df.actual_label.values, df.model_LR.values)
```
[16] ✓ 0.0s

```
import matplotlib.pyplot as plt

plt.plot(fpr_RF, tpr_RF, "r-", label="RF")
plt.plot(fpr_LR, tpr_LR, "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()
```
[17] ✓ 0.2s

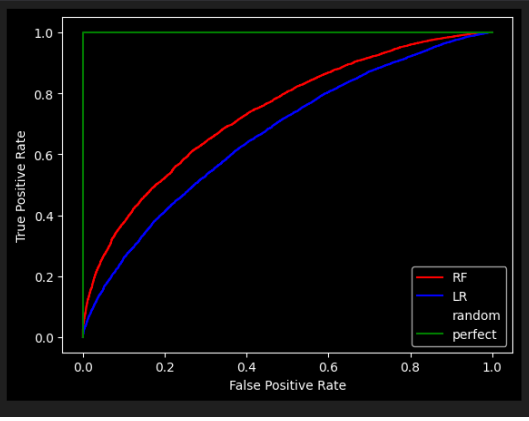

Рис. 6.9 – Результат виконання програми

```
from sklearn.metrics import roc_auc_score

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)
```
[18] ✓ 0.0s

```
AUC RF:0.738
AUC LR:0.666
```

```
import matplotlib.pyplot as plt

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()
```
[19] ✓ 0.2s

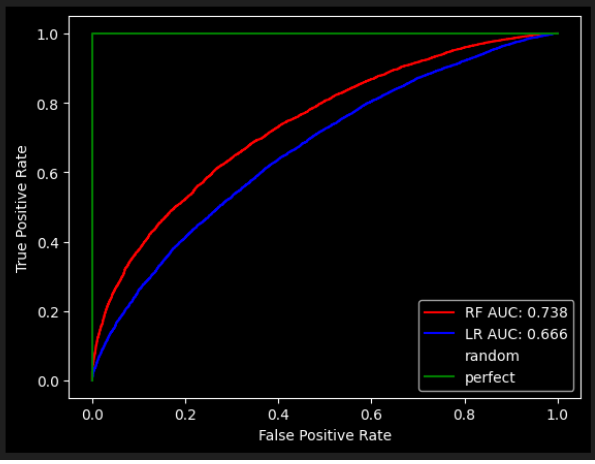

Рис. 6.10 – Результат виконання програми

Модель RF краще за LR, бо вона більше наближена до ідеальної кривої.

Завдання 7: 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

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
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 класифікатора машини опорних векторів
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 SVM classifier =", round(accuracy, 2), "%")

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

[1] ✓ 13.4s

```
Accuracy of SVM classifier = 99.75 %
```

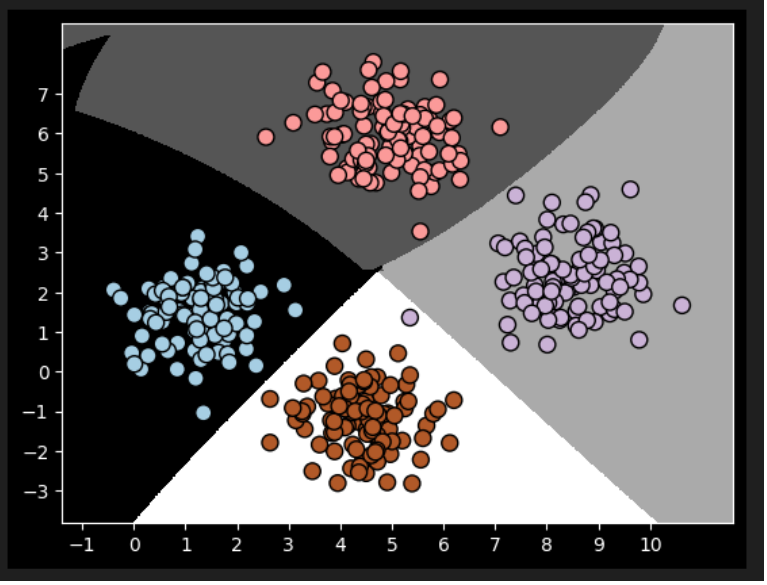

Рис. 7.1 – Результат виконання програми

```python
# 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="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)) + "%")
```

[2] ✓ 9.9s

```
Accuracy of the new classifier = 100.0 %
```

Рис. 7.2 – Результат виконання програми

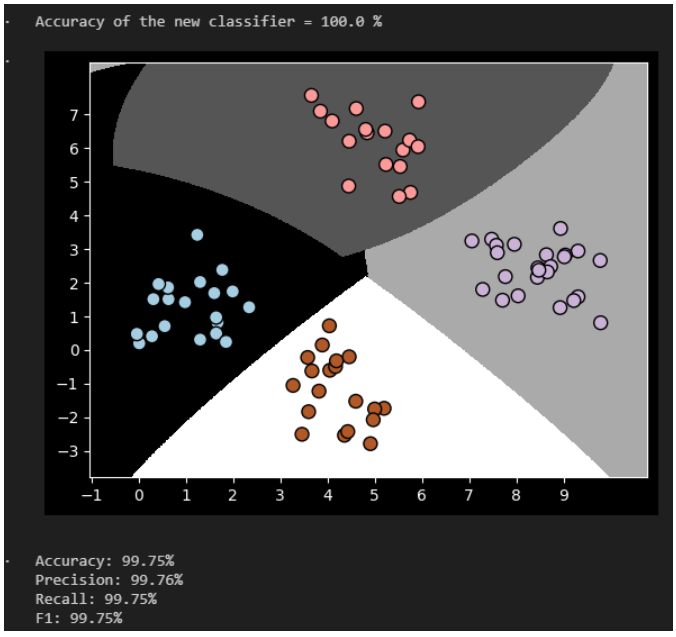

Рис. 7.3 – Результат виконання програми

Висновок: в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python досліджено попередню обробку та класифікацію даних. Отримано результати в числовому та графічному вигляді.