ЛАБОРАТОРНА РОБОТА № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

LR_1_task_1.py

```python
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)

# 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))
```

```
Label mapping:
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)]

Encoded values = [3, 0, 4, 1]
Decoded labels = [np.str_('white'), np.str_('Black'), np.str_('yellow'), np.str_('green')]
```

LR_1_task_2.py

```python
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, 3.1, 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))

# 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. 0. 0.]
 [0. 0. 1.]
 [1. 1. 1.]
 [0. 0. 0.]]

BEFORE:
Mean = [-1.475  0.425  0.775]
Std deviation = [2.82610598 1.92662269 4.79446295]

AFTER:
Mean = [-5.55111512e-17 -5.55111512e-17 -4.16333634e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.27631579 0.         0.        ]
 [0.26315789 0.08510638 0.85245902]
 [1.         1.         1.        ]
 [0.         0.63829787 0.40163934]]

l1 normalized data:
 [[-0.23       -0.16       -0.61      ]
 [-0.30379747 -0.15189873  0.5443038 ]
 [ 0.25806452  0.25        0.49193548]
 [-0.62857143  0.2        -0.17142857]]

l2 normalized data:
 [[-0.34263541 -0.23835507 -0.90872869]
 [-0.47351004 -0.23675502  0.84837215]
 [ 0.42362745  0.41038909  0.80753983]
 [-0.92228798  0.29345527 -0.25153308]]
```

LR_1_task_3.py

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

```python
# 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)
```

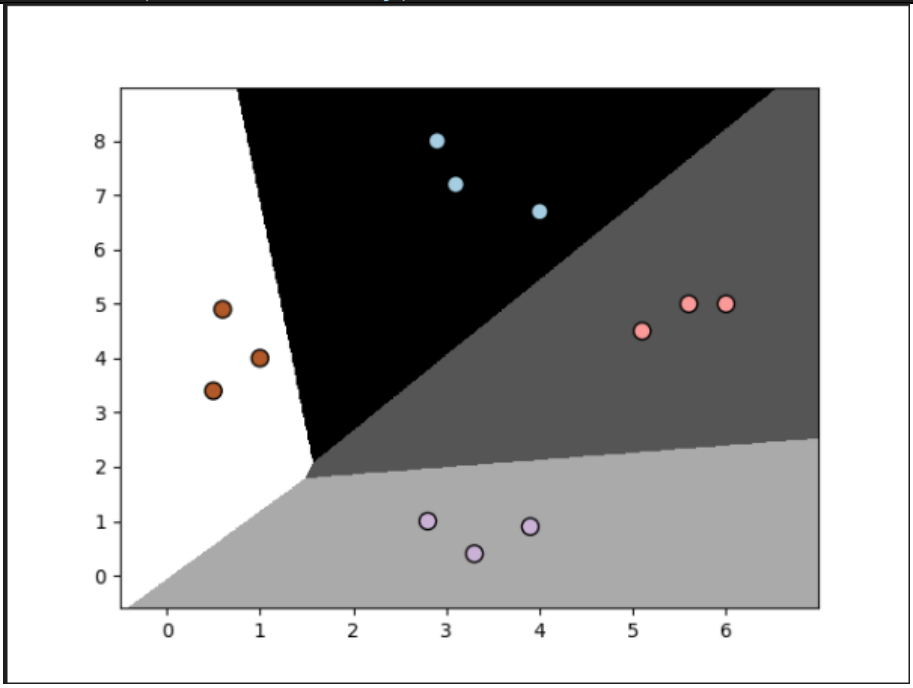

Рис. 4 результат роботи класифікатора

LR_1_task_4.py

```python
import numpy as np
```

```python
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.naive_bayes import GaussianNB

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, "task4.1")


# 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)
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, "task4.2")

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)) + "%")
```

```
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%
```

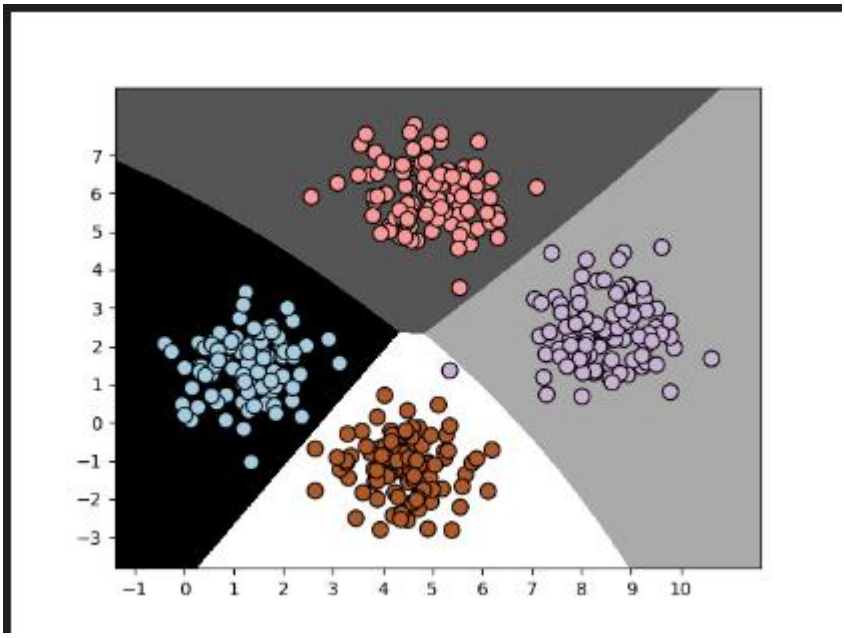

Рис. 5 – Без перехресної перевірки

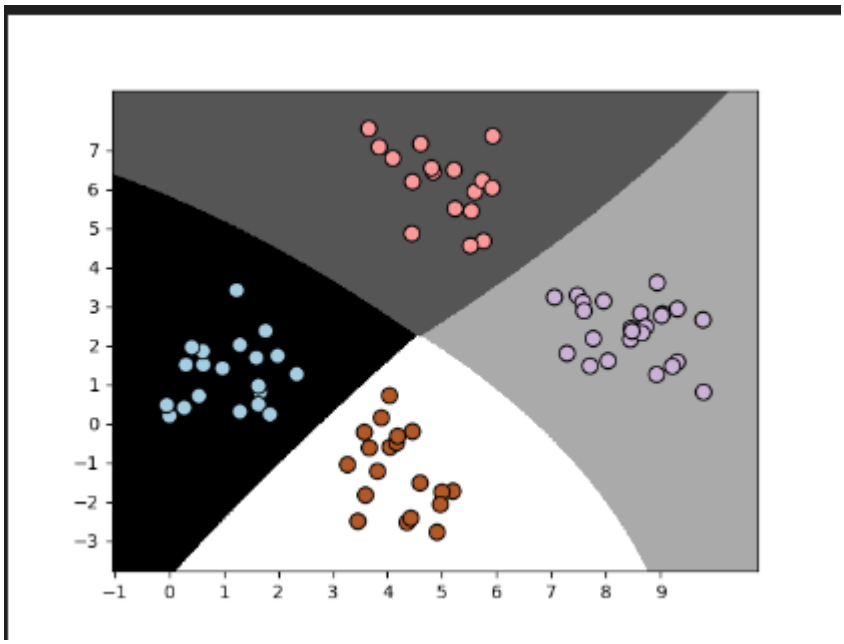

Рис. 6 – З перехресною перевіркою

Завдяки перехресній перевірці результати вийшли більш точними.

LR_1_task_5.py

```python
import numpy as np
import pandas as pd
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    f1_score,
    precision_score,
    recall_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()

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


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

assert np.array_equal(
    ozornin_confusion_matrix(df.actual_label.values, df.predicted_RF.values),
    confusion_matrix(df.actual_label.values, df.predicted_RF.values),
), "ozornin_confusion_matrix() is not correct for RF"
assert np.array_equal(
    ozornin_confusion_matrix(df.actual_label.values, df.predicted_LR.values),
    confusion_matrix(df.actual_label.values, df.predicted_LR.values),
), "ozornin_confusion_matrix() is not correct for LR"


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


def ozornin_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 + FN + FP + TN)


assert ozornin_accuracy_score(
    df.actual_label.values, df.predicted_RF.values
) == accuracy_score(
    df.actual_label.values, df.predicted_RF.values
), "my_accuracy_score failed on RF"
assert ozornin_accuracy_score(
    df.actual_label.values, df.predicted_LR.values
) == accuracy_score(
    df.actual_label.values, df.predicted_LR.values
), "my_accuracy_score failed on LR"
print(
    "Accuracy RF:%.3f"
    % (ozornin_accuracy_score(df.actual_label.values, df.predicted_RF.values))
)

print(
    "Accuracy LR: %.3f"
    % (ozornin_accuracy_score(df.actual_label.values, df.predicted_LR.values))
)


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

```python
def ozornin_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 ozornin_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 ozornin_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"
    % (ozornin_recall_score(df.actual_label.values, df.predicted_RF.values))
)
print(
    "Recall LR: %.3f"
    % (ozornin_recall_score(df.actual_label.values, df.predicted_LR.values))
)


def ozornin_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 ozornin_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 ozornin_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"
    % (ozornin_precision_score(df.actual_label.values, df.predicted_RF.values))
)
print(
    "Precision LR: %.3f"
    % (ozornin_precision_score(df.actual_label.values, df.predicted_LR.values))
)
```

```python
def ozornin_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = ozornin_recall_score(y_true, y_pred)
    precision = ozornin_precision_score(y_true, y_pred)
    return (2 * precision * recall) / (precision + recall)


assert ozornin_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"

print(
    "F1 RF: %.3f" % (ozornin_f1_score(df.actual_label.values,
df.predicted_RF.values))
)
print(
    "Ozornin F1 LR: %.3f"
    % (ozornin_f1_score(df.actual_label.values, df.predicted_LR.values))
)
print("Lib F1 LR: %.3f" % (f1_score(df.actual_label.values,
df.predicted_LR.values)))

# assert ozornin_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("scores with threshold = 0.5")

print(
    "Accuracy RF: %.3f"
    % (ozornin_accuracy_score(df.actual_label.values, df.predicted_RF.values))
)
print(
    "Recall RF: %.3f"
    % (ozornin_recall_score(df.actual_label.values, df.predicted_RF.values))
)
print(
    "Precision RF: %.3f"
    % (ozornin_precision_score(df.actual_label.values, df.predicted_RF.values))
)
print(
    "F1 RF: %.3f" % (ozornin_f1_score(df.actual_label.values,
df.predicted_RF.values))
)
print("")
print("scores with threshold = 0.25")
print(
    "Accuracy RF: %.3f"
    % (
        ozornin_accuracy_score(
            df.actual_label.values, (df.model_RF >= 0.25).astype("int").values
        )
```

```
    )
)
print(
    "Recall RF: %.3f"
    % (
        ozornin_recall_score(
            df.actual_label.values, (df.model_RF >= 0.25).astype("int").values
        )
    )
)
print(
    "Precision RF: %.3f"
    % (
        ozornin_precision_score(
            df.actual_label.values, (df.model_RF >= 0.25).astype("int").values
        )
    )
)
print(
    "F1 RF: %.3f"
    % (
        ozornin_f1_score(
            df.actual_label.values, (df.model_RF >= 0.25).astype("int").values
        )
    )
)

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)
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.savefig("curve1.png")

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)
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.savefig("curve2.png")
```

```
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
Ozornin F1 LR: 0.586
Lib 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
```

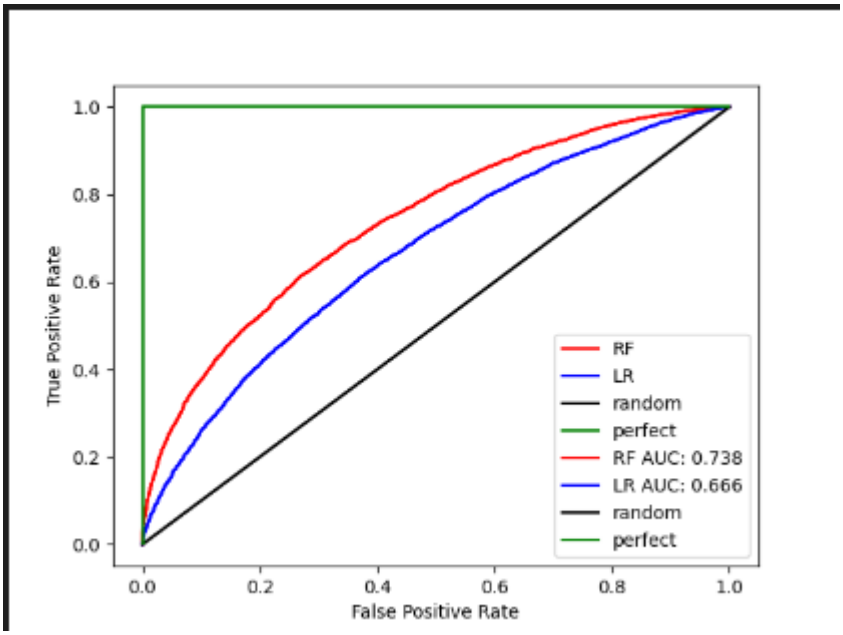

Рис. 7 – ROC(AUC) кожної моделі

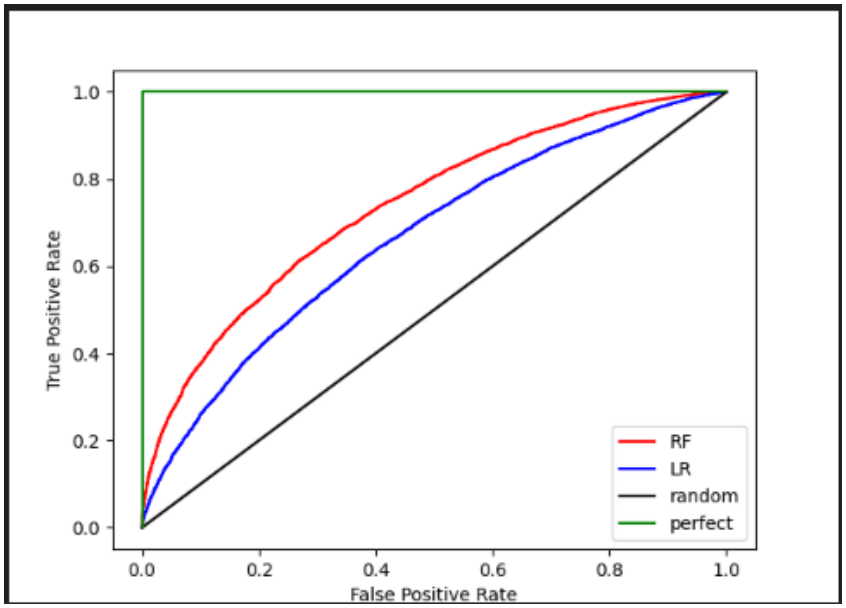

Рис. 8 – ROC кожної моделі

Обидві моделі показують однакові результати, тому вони однаково ефективні

LR_1_task_6.py

```
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

# Завантажимо дані
data = pd.read_csv(
    "data_multivar_nb.txt", sep=","
)  # Замість ',' вкажіть коректний роздільник
X = data.iloc[:, :-1].values  # Ознаки
y = data.iloc[:, -1].values  # Цільові значення

# Розділимо на тренувальні та тестові дані
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=3)

# Класифікація за допомогою SVM
svm_classifier = SVC(kernel="linear", random_state=42)
svm_classifier.fit(X_train, y_train)

# Прогнозуємо на тестових даних
y_pred_svm = svm_classifier.predict(X_test)

# Розраховуємо метрики якості
print("SVM Classifier Metrics:")
```

```python
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Classification Report:\n", classification_report(y_test, y_pred_svm))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))

# Класифікація за допомогою наївного Байєсівського класифікатора
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

# Прогнозуємо на тестових даних
y_pred_nb = nb_classifier.predict(X_test)

# Розраховуємо метрики якості
print("\nNaive Bayes Classifier Metrics:")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_nb))
```

```
SVM Classifier Metrics:
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      1.00      1.00        17
           2       1.00      1.00      1.00        14
           3       1.00      1.00      1.00        28
    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80

Confusion Matrix:
 [[21  0  0  0]
 [ 0 17  0  0]
 [ 0  0 14  0]
 [ 0  0  0 28]]

Naive Bayes Classifier Metrics:
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      1.00      1.00        17
           2       1.00      1.00      1.00        14
           3       1.00      1.00      1.00        28
    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80

Confusion Matrix:
 [[21  0  0  0]
 [ 0 17  0  0]
 [ 0  0 14  0]
 [ 0  0  0 28]]
```

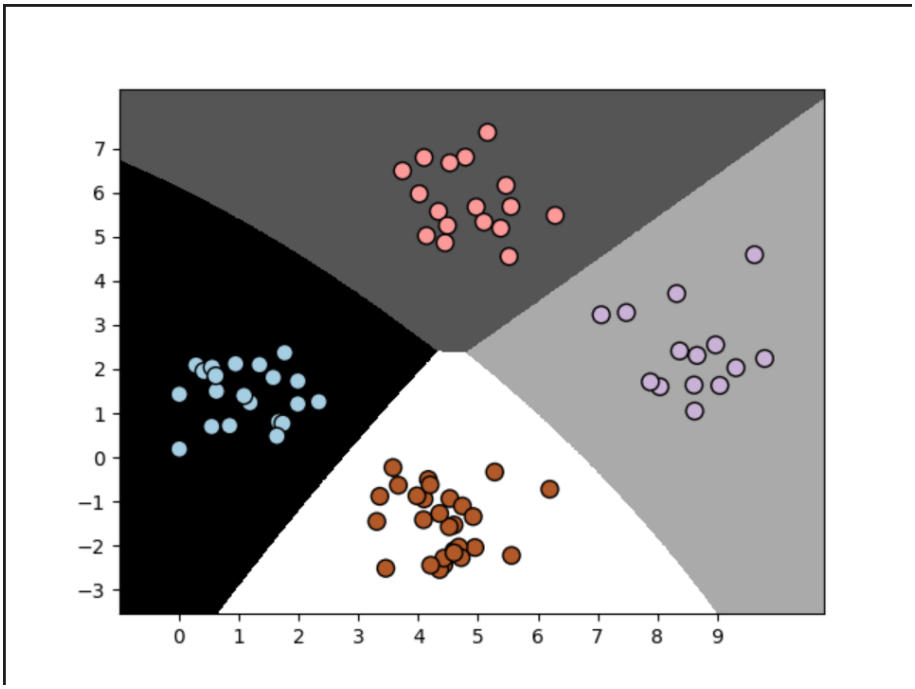

Рис. 9 – Наївний Байєс

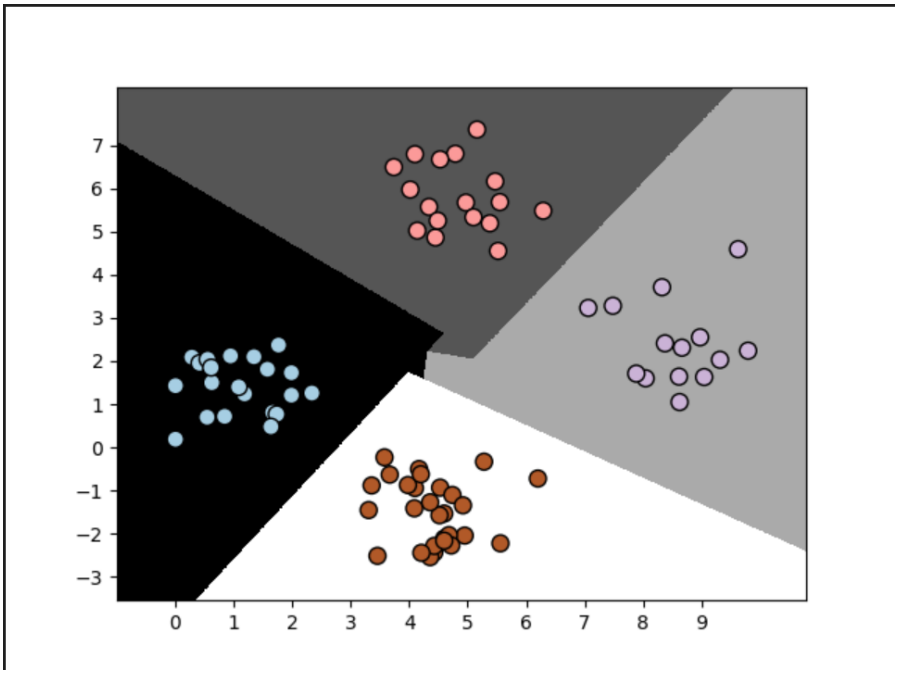

Рис. 10 – SVM

Показники якості класифікацій є ідентичними. Однак і зміни у графіках є. Змінились області значень для кожної класифікації. Тому в нашому випадку Модель SVM є кращою.

GitHub = https://github.com/Ozzornin/AIS/tree/master