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

Посилання на GitHub: https://github.com/Max2002/AI_IPZ-19-3_LMV

Завдання 1. Нормалізація даних. Кодування міток.

Лістинг файлу LR_1_task_1.py:

```
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]])

# 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)

# 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))
```

```
C:\Users\Max\AppData\Local\Programs\Python\Python311\python.exe "E:\Лабораторні\4 курс\Системи штучного інтелекту\Lab - 1\project\LR_1_task_1.py"

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]]

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

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

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

Process finished with exit code 0
```

Рис. 1. Кодування міток.

Порівняння нормалізації L1 та L2: Нормалізація L2 має результат в цілих числах, тому вона не така точна в порівняні з нормалізацією L1, але L1 не дозволяє вирішувати завдання, де необхідно простежувати неточність вхідних даних (викиди).

Завдання 2. Попередня обробка нових даних.

Лістинг файлу LR_1_task_2.py:

```python
import numpy as np
from sklearn import preprocessing

input_data = np.array([[-5.3, -8.9, 3.0],
                       [2.9, 5.1, -3.3],
                       [3.1, -2.8, -3.2],
                       [2.2, -1.4, 5.1]])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=3.0).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)
```

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

BEFORE:
Mean = [ 0.725 -2.     0.4  ]
Std deviation = [3.49454933 4.97543968 3.72491611]

AFTER:
Mean = [-2.77555756e-17 -2.42861287e-17  0.00000000e+00]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.         0.         0.75      ]
 [0.97619048 1.         0.        ]
 [1.         0.43571429 0.01190476]
 [0.89285714 0.53571429 1.        ]]

l1 normalized data:
 [[-0.30813953 -0.51744186  0.1744186 ]
 [ 0.25663717  0.45132743 -0.2920354 ]
 [ 0.34065934 -0.30769231 -0.35164835]
 [ 0.25287356 -0.16091954  0.5862069 ]]

l2 normalized data:
 [[-0.49145755 -0.82527777  0.27818352]
 [ 0.43082507  0.75765788 -0.49024922]
 [ 0.58911518 -0.53210404 -0.6081189 ]
 [ 0.38407812 -0.24441335  0.89036291]]
```

Рис. 2. Результат виконання завдання.

Завдання 3. Класифікація логістичною регресією або логістичний класифікатор.

Лістинг файлу LR_1_task_3.py:

```
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)
```

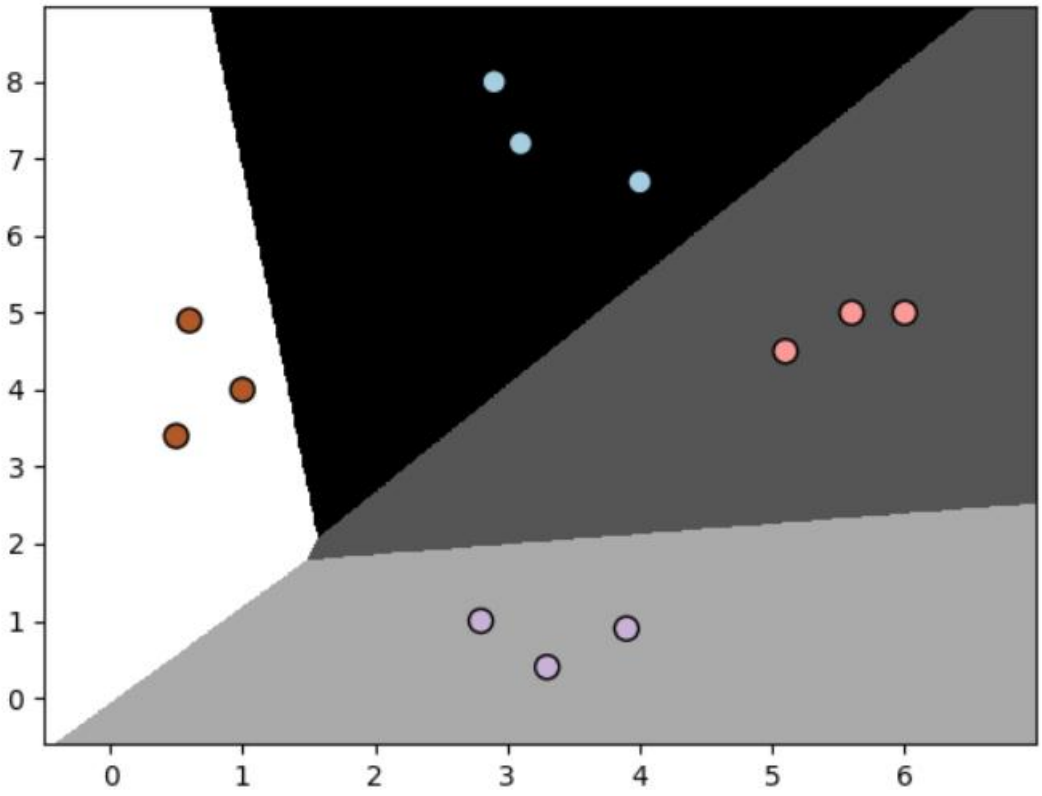

Рис. 3. Результат класифікації лінійною регресією.

Завдання 4. Класифікація наївним байєсовським класифікатором.

Лістинг файлу LR_1_task_4.py:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
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 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)

# 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)) + "%")
```

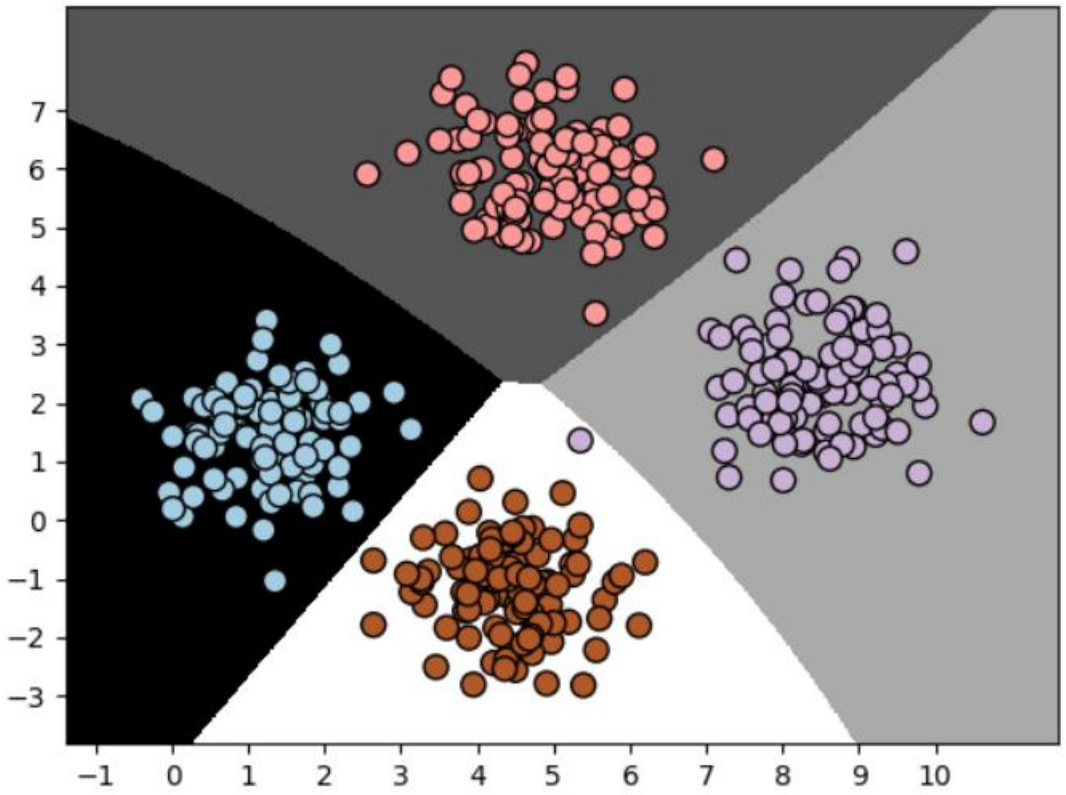

Рис. 4. Відображення результату класифікації

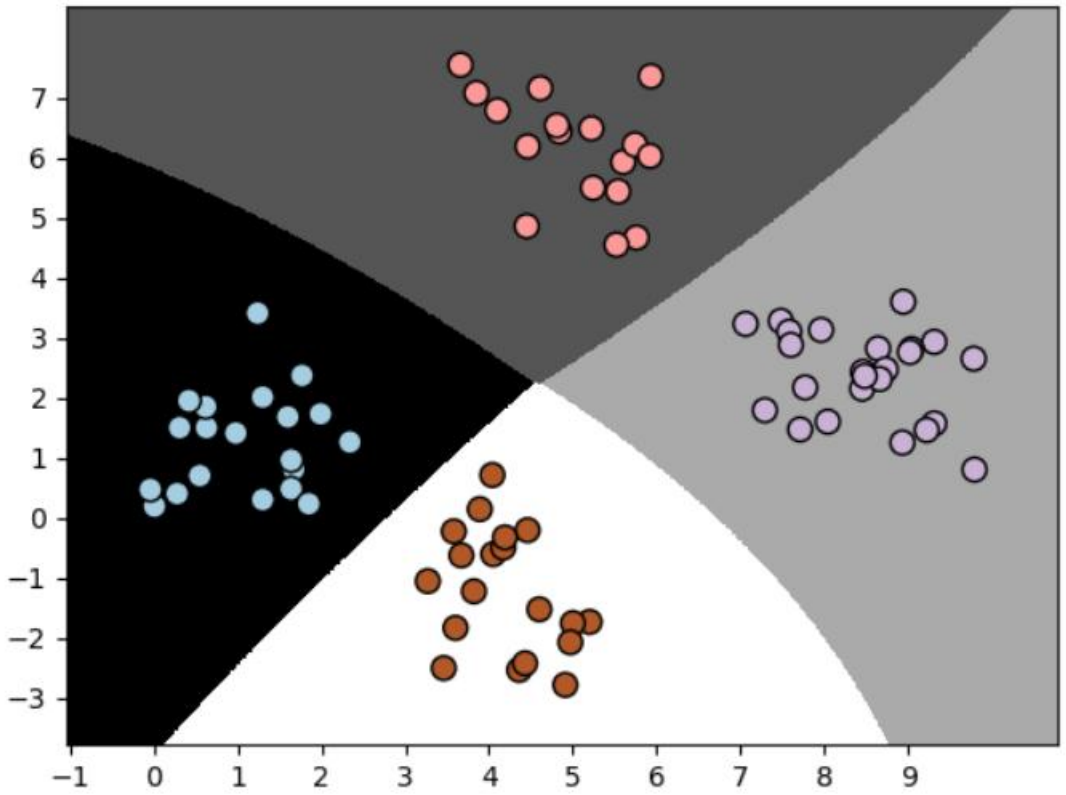

Рис. 5. Зображення результату класифікації тестових даних

```
C:\Users\Max\AppData\Local\Programs\Python\Python311\python.exe "E:\Лабораторні\4 курс\Системи штучного інтелекту\Lab - 1\project\LR_1_task_4.py"
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. Дані про якість класифікатора

Завдання 5. Вивчити метрики якості класифікації

Лістинг файлу LR_1_task_5.py:

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

df = pd.read_csv('data_metrics.csv')
df.head()

thresh = 0.5
df['predicted_RF'] = (df.model_RF >= thresh).astype('int')
df['predicted_LR'] = (df.model_LR >= thresh).astype('int')
df.head()

# confusion_matix

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


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


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


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


def find_TN(y_true, y_pred):
    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):
    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 liashuk_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(liashuk_confusion_matrix(df.actual_label.values, df.predicted_RF.values))

assert np.array_equal(liashuk_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(liashuk_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'


# accuracy
score = accuracy_score(df.actual_label.values, df.predicted_RF.values)
print("Accuracy score on RF:", score)


def liashuk_accuracy_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return (TP + TN) / (TP + FN + FP + TN)


assert liashuk_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 liashuk_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("my accuracy score on RF:", liashuk_accuracy_score(df.actual_label.values,
df.predicted_RF.values))
print("my accuracy score on LR:", liashuk_accuracy_score(df.actual_label.values,
df.predicted_LR.values))

# Recall
print('Recall score on RF:', recall_score(df.actual_label.values,
df.predicted_RF.values))


def liashuk_recal_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FN)


assert liashuk_recal_score(df.actual_label.values, df.predicted_RF.values) ==
recall_score(df.actual_label.values, df.predicted_RF.values),\
    'my recal_score fails on RF'

assert liashuk_recal_score(df.actual_label.values, df.predicted_LR.values) ==
recall_score(df.actual_label.values, df.predicted_LR.values),\
    'my recal_score fails on LR'

print("My recall score on RF:", liashuk_recal_score(df.actual_label.values,
df.predicted_RF.values))
print("My recall score on LR:", liashuk_recal_score(df.actual_label.values,
```

```
df.predicted_LR.values))

# precision_score

print("Precision score on RF:", precision_score(df.actual_label.values,
df.predicted_RF.values))

def liashuk_precision_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FP)


assert liashuk_precision_score(df.actual_label.values, df.predicted_RF.values) ==
precision_score(df.actual_label.values, df.predicted_RF.values),\
    'my precision_score fails on RF'

assert liashuk_precision_score(df.actual_label.values, df.predicted_LR.values) ==
precision_score(df.actual_label.values, df.predicted_LR.values),\
    'my precision_score fails on LR'

print("my precision score on RF:", liashuk_precision_score(df.actual_label.values,
df.predicted_RF.values))
print("my precision score on LR:", liashuk_precision_score(df.actual_label.values,
df.predicted_LR.values))

# F1 score
print("F1 score on RF", f1_score(df.actual_label.values, df.predicted_RF.values))
def liashuk_f1_score(y_true, y_pred):
    precision = liashuk_precision_score(y_true, y_pred)
    recall = liashuk_recal_score(y_true, y_pred)
    return (2 * (precision * recall)) / (precision + recall)


assert liashuk_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values, df.predicted_RF.values),\
    'my f1_score fails on RF'

assert liashuk_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values, df.predicted_LR.values),\
    'my f1_score fails on LR'

print("My F1 score score on RF:", liashuk_f1_score(df.actual_label.values,
df.predicted_RF.values))
print("My F1 score score on LR:", liashuk_f1_score(df.actual_label.values,
df.predicted_LR.values))
print()


def test_thresholds(threshold: float):
    print(f"Scores with threshold = {threshold}")
    predicted = (df.model_RF >= threshold).astype('int')

    print("Accuracy RF:", liashuk_accuracy_score(df.actual_label.values,
predicted))
    print("Precision RF:", liashuk_precision_score(df.actual_label.values,
predicted))
    print("Recall RF:", liashuk_recal_score(df.actual_label.values, predicted))
    print("F1 RF:", liashuk_f1_score(df.actual_label.values, predicted))
    print()


test_thresholds(thresh)
test_thresholds(.25)
```

```
test_thresholds(.75)
test_thresholds(.15)

# 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)

# 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:", auc_RF)
print("AUC LR:", auc_LR)

plt.plot(fpr_RF, tpr_RF, 'r-', label=f'AUC RF: {auc_RF}')
plt.plot(fpr_LR, tpr_LR, 'b-', label=f'AUC LR: {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()
```

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

Рис. 7. Результат роботи функції по обчисленю матриці помилок(результат збігається)

```
Accuracy score on RF: 0.6705165630156111
my accuracy score on RF: 0.6705165630156111
my accuracy score on LR: 0.6158141896179719
Recall score on RF: 0.6405635232897576
My recall score on RF: 0.6405635232897576
My recall score on LR: 0.5430892245208783
Precision score on RF: 0.681382476036182
my precision score on RF: 0.681382476036182
my precision score on LR: 0.6355265112134264
F1 score on RF 0.660342797330891
My F1 score score on RF: 0.660342797330891
My F1 score score on LR: 0.5856830002737475
```

Рис. 8. Метрика моделей. Перевірка результатів власних функцій

```
Scores with threshold = 0.5
Accuracy RF: 0.6705165630156111
Precision RF: 0.681382476036182
Recall RF: 0.6405635232897576
F1 RF: 0.660342797330891

Scores with threshold = 0.25
Accuracy RF: 0.5024114735372509
Precision RF: 0.5012086513994911
Recall RF: 1.0
F1 RF: 0.6677401584812916

Scores with threshold = 0.75
Accuracy RF: 0.5123746668358928
Precision RF: 0.9949238578680203
Recall RF: 0.02487625333164107
F1 RF: 0.04853888063397722
```

Рис. 9. Метрика моделі RF за різних порогів

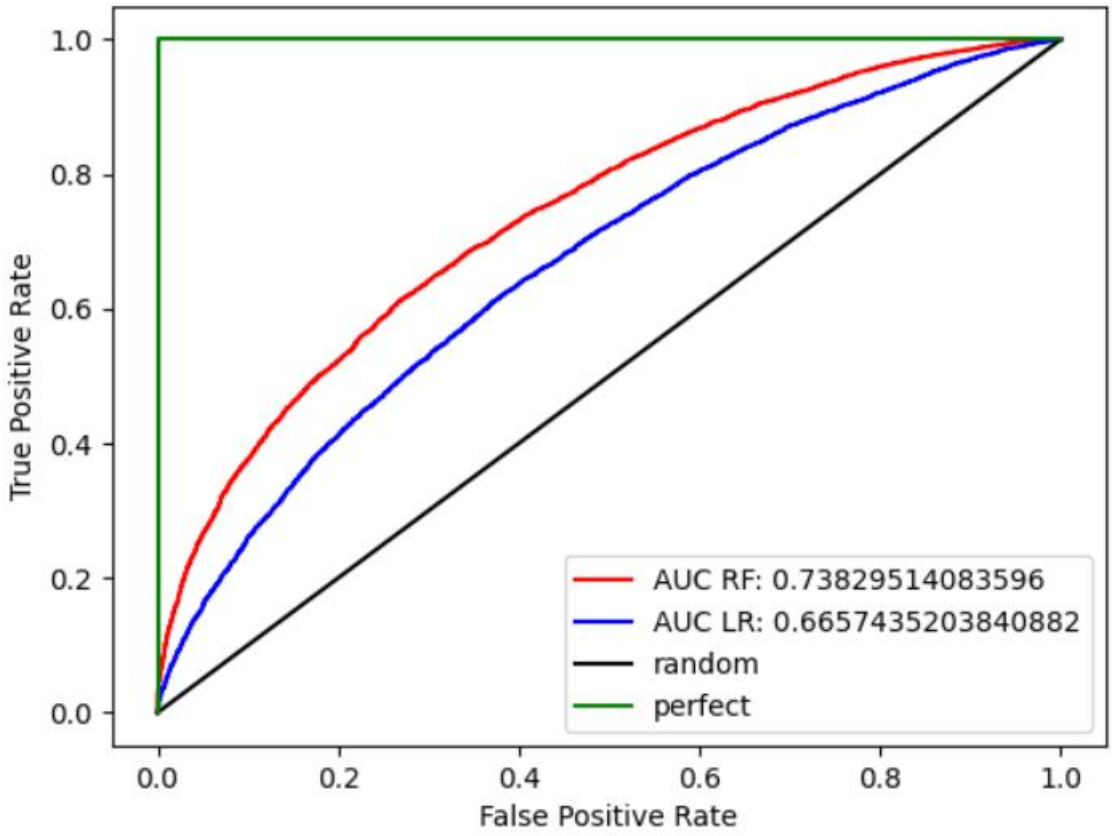

Рис. 10. Графік отриманих значень ROC

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 коду файлу LR_1_task_6.py:

```
import numpy as np
from sklearn import svm
from sklearn.model_selection import train_test_split, cross_val_score
from utilities import visualize_classifier

input_file = 'data_multivar_nb.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=3)
classifier = svm.SVC(decision_function_shape='ovr')
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)

visualize_classifier(classifier, X_test, y_test)
```

```
num_folds = 3
accuracy_values = cross_val_score(classifier, X_test, y_test, scoring='accuracy',
cv=num_folds)
print(f"Accuracy: {round(100 * accuracy_values.mean(), 2)}%")

precision_values = cross_val_score(classifier, X_test, y_test,
scoring='precision_weighted', cv=num_folds)
print(f"Precision: {round(100 * precision_values.mean(), 2)}%")

recall_values = cross_val_score(classifier, X_test, y_test,
scoring='recall_weighted', cv=num_folds)
print(f"Recall: {round(100 * recall_values.mean(), 2)}%")

f1_values = cross_val_score(classifier, X_test, y_test, scoring='f1_weighted',
cv=num_folds)
print(f"F1: {round(100 * f1_values.mean(), 2)}%")
```

```
C:\Users\Max\AppData\Local\Programs\Python\Python311\python.exe "E:\Лабораторні\4 курс\Системи штучного інтелекту\Lab - 1\project\LR_1_task_6.py"
Accuracy: 100.0%
Precision: 100.0%
Recall: 100.0%
F1: 100.0%
```

Рис. 11. Показники якості класифікатора

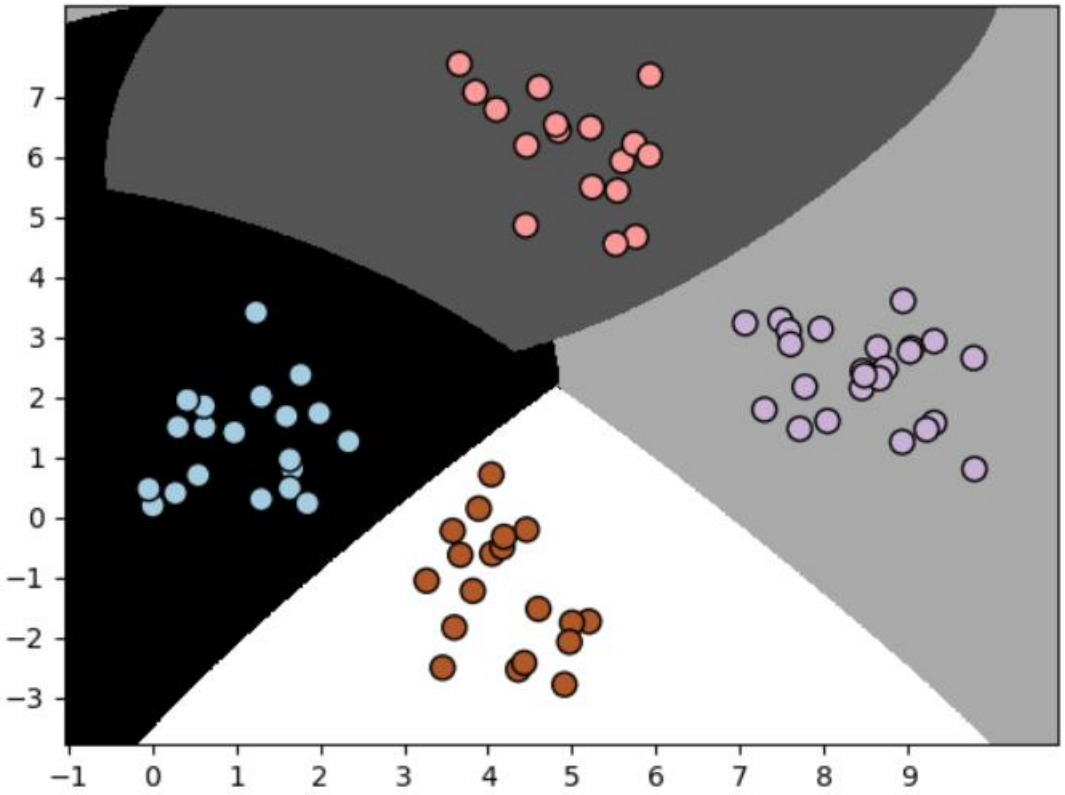

Рис. 12. Результат класифікації тестових даних за допомоги SVM

Висновки по використанню SVM класифікатора в порівняні з байєсівським класифікатором: SVM класифікатор є швидшим та простішим, але для

використання у багатокласовій класифікації не пристосований. Окрім цього, кількість даних може бути недостатньою через однакові показники.

Висновок: було досліджено попередню обробку та класифікацію даних, використовуючи спеціальні бібліотеки та мову програмування Python.