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

Для роботи з даними необхідно використовувати спеціалізовані бібліотеки функцій. Надалі використовуються numpy та sklearn.

Лістинг коду підключень бібліотек файлу Task1.py:

```
1    import numpy as np
2    from sklearn import preprocessing
```

Серед методів попередньої обробки даних досліджуються бінаризація, виключення середнього, масштабування, нормалізація.

Лістинг коду методів обробки файлу Task1.py

```
# Data
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(f"Binarized data:\n{data_binarized}")

# Виключення середнього
print("\nBefore:")
print("Mean = ", input_data.mean(axis=0))
print("Std deviation = ", input_data.std(axis=0))

data_scaled = preprocessing.scale(input_data)
print("\nAfter:")
print("Mean = ", data_scaled.mean(axis=0))
print("Std deviation = ", data_scaled.std(axis=0))
```

```
# Масштабування
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
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("l2 normalized data:\n", data_normalized_l2)
```

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

Before:
Mean =  [-4.075  1.05   2.675]
Std deviation =  [1.47542367 4.40028408 2.88823043]

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

Рис. 1.1 – Бінаризація та виключення середнього

```
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]]
```

Рис.1.2 – Масштабування та нормалізація власних даних

Нормалізація L1 та L2 відрізняються точністю значень, отриманих в розрахунках суми (абсолютних значень за L1 та квадратів значень за L2). Застосування 2-го методу надає меншу точність та є менш надійним, у той час як

1-й не дозволяє вирішувати завдання, де необхідно простежувати неточність вхідних даних (викиди).

Для класифікації даних необхідно працювати з мітками, які часто для зручності є текстовими. Використовувані функції машинного навчання передбачають використання чисельних міток, через що необхідно текстові мітки перетворювати, використовуючи їх кодування.

Лістинг коду кодування міток файлу Task1.py:

```
## Кодування міток
input_labels = ['red', 'black', 'red', 'green', 'black', 'yellow', 'white']
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)
print("\nLabel mapping:")
for i, item in enumerate(encoder.classes_):
    print(f"{item} --> {i}")

test_labels = ['green', 'red', 'black']
encoded_values = encoder.transform(test_labels)
print("\nLabels: ", test_labels)
print("Encoded values: ", encoded_values)

encoded_values = [3, 0, 4, 1]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values: ", encoded_values)
print("Decoded labels: ", decoded_list)
```

```
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']
```

Рис.1.3 – Кодування міток

Завдання 2. Попередня обробка нових даних

Необхідно виконати операції бінаризації, виключення середнього, масштабування та нормалізації відносно нових даних власного варіанту (11й).

Лістинг коду файлу Task_2.py:

```
import numpy as np
from sklearn import preprocessing

# Дані до обробки (22й варіант)
```

```python
input_data = np.array([
    [-1.6, 3.9, 4.5],
    [-4.3, 4.2, 3.3],
    [-5.2, -6.5, 5.1],
    [-5.2, 2.6, -2.2]
])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=3.8).transform(input_data)
print(f"Binarized data:\n{data_binarized}")

# Виключення середнього
print("\nBefore:")
print("Mean = ", input_data.mean(axis=0))
print("Std deviation = ", input_data.std(axis=0))

data_scaled = preprocessing.scale(input_data)
print("\nAfter:")
print("Mean = ", data_scaled.mean(axis=0))
print("Std deviation = ", data_scaled.std(axis=0))

# Масштабування
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
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("l2 normalized data:\n", data_normalized_l2)
```

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

Before:
Mean =  [-4.075  1.05   2.675]
Std deviation =  [1.47542367 4.40028408 2.88823043]

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

Рис.1.4 – Бінаризація та виключення середнього власних даних

```
Min max scaled data:
 [[1.         0.97196262 0.91780822]
 [0.25       1.         0.75342466]
 [0.         0.         1.        ]
 [0.         0.85046729 0.        ]]

l1 normalized data:
 [[-0.16        0.39        0.45      ]
 [-0.36440678  0.3559322   0.27966102]
 [-0.30952381 -0.38690476  0.30357143]
 [-0.52        0.26       -0.22      ]]
l2 normalized data:
 [[-0.259486    0.63249712  0.72980437]
 [-0.62708606  0.61250266  0.48125209]
 [-0.53266835 -0.66583544  0.52242473]
 [-0.83653629  0.41826814 -0.3539192 ]]
```

Рис.1.5 – Масштабування та нормалізація власних даних

Завдання 3. Класифікація логістичною регресією або логістичний класифікатор

Для класифікації даних, а саме спрощення цього, використовується логістична регресія. Завдяки модулю utilities.py, який було надано для виконання лабораторної роботи,

Лістинг коду файлу Task_3.py:

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


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])

classifier = linear_model.LogisticRegression(solver="liblinear", C=1)
classifier.fit(X, y)
visualize_classifier(classifier, X, y)
```

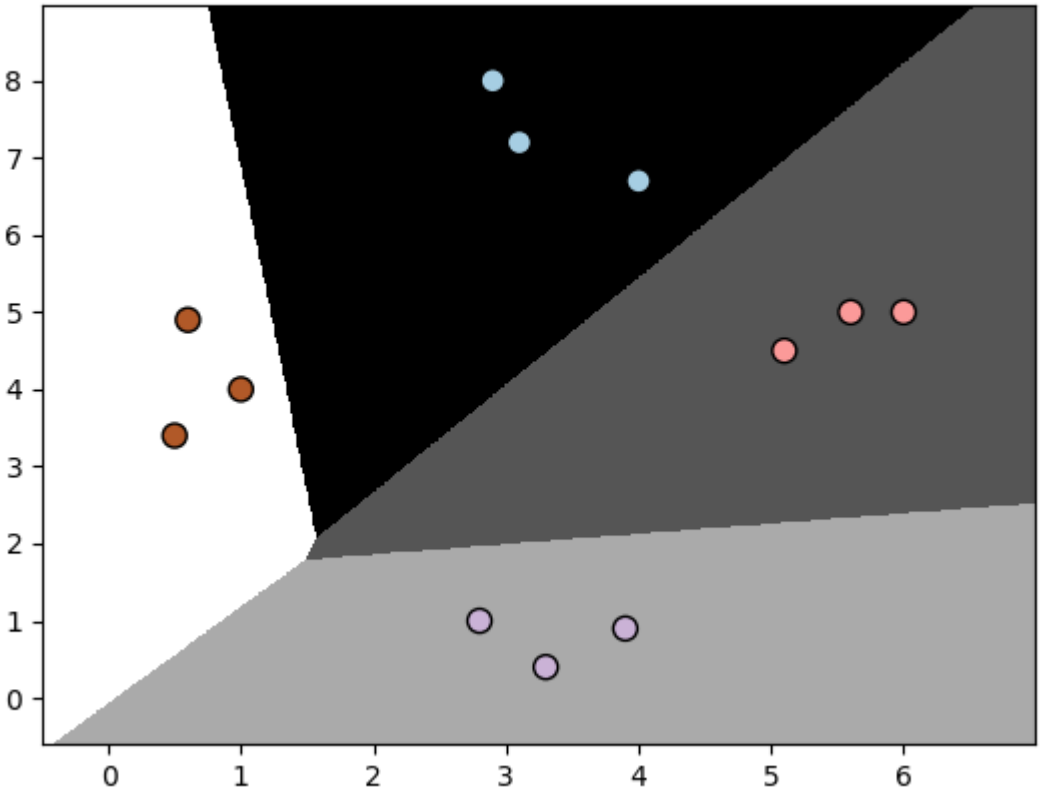

Рис.1.6 – Результат класифікації лінійною регресією

Завдання 4. Класифікація наївним байєсовським класифікатором

Наївний Байєс є набором методів класифікації, що не бере до уваги можливість залежності ознак між собою та наразі існує лише як навчальний приклад.

Лістинг коду файлу Task_4.py:

```
import numpy as np
from sklearn.naive_bayes import GaussianNB
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]

classifier = GaussianNB()
classifier.fit(X, y)
y_pred = classifier.predict(X)

accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]
print(f"Accuracy of Naive Bayes classifier: {round(accuracy, 2)}%")
visualize_classifier(classifier, X, y)
```

```
Accuracy of Naive Bayes classifier: 99.75%
```

Рис.1.7 – Якість класифікатора

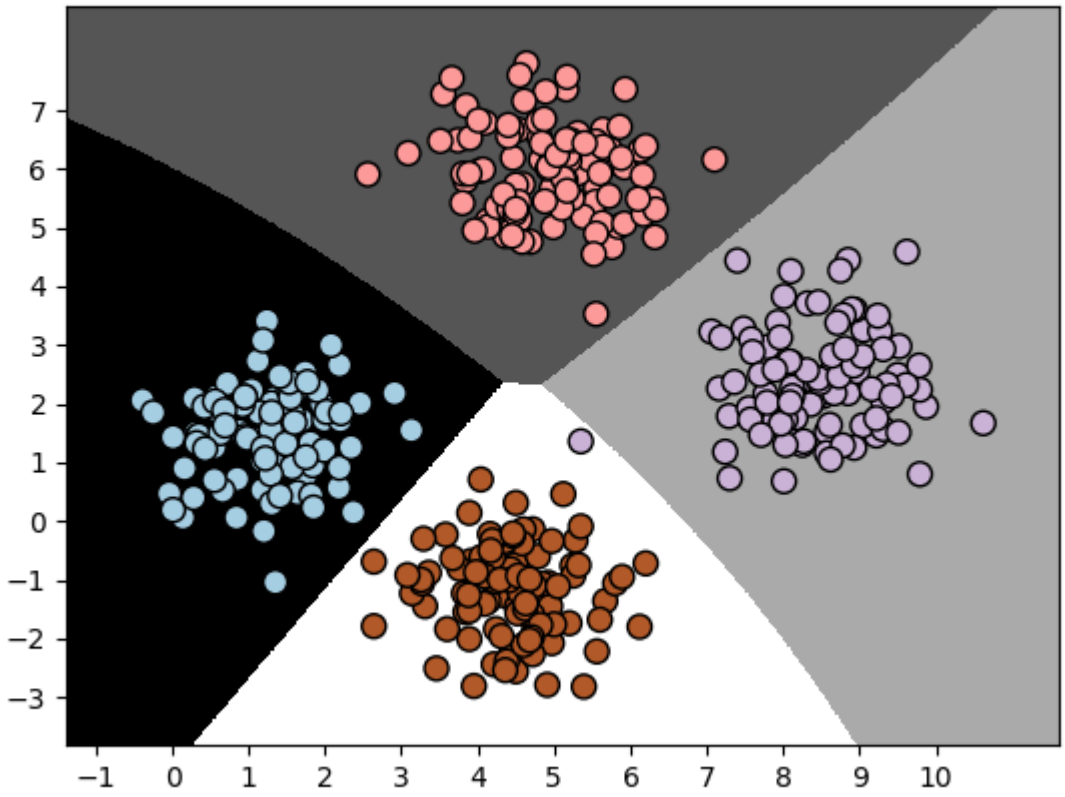

Рис.1.8 – Відображення результату класифікації

Для якісного обрахунку точності необхідно розділити дані на навчальний та тесовий набори.

Лістинг коду файлу Task_4.py з розділенням даних:

```
# Аналіз із розділенням 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)

accuracy = 100 * (y_test == y_test_pred).sum() / X_test.shape[0]
print(f"Accuracy of the new Naive Bayes classifier: {round(accuracy, 2)}%")
visualize_classifier(classifier_new, X_test, y_test)

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

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

```
Accuracy of the new Naive Bayes classifier: 100.0%
Accuracy: 100.0%
Precision: 100.0%
Recall: 100.0%
F1: 100.0%
```

Рис.1.9 – Отримані дані про якість

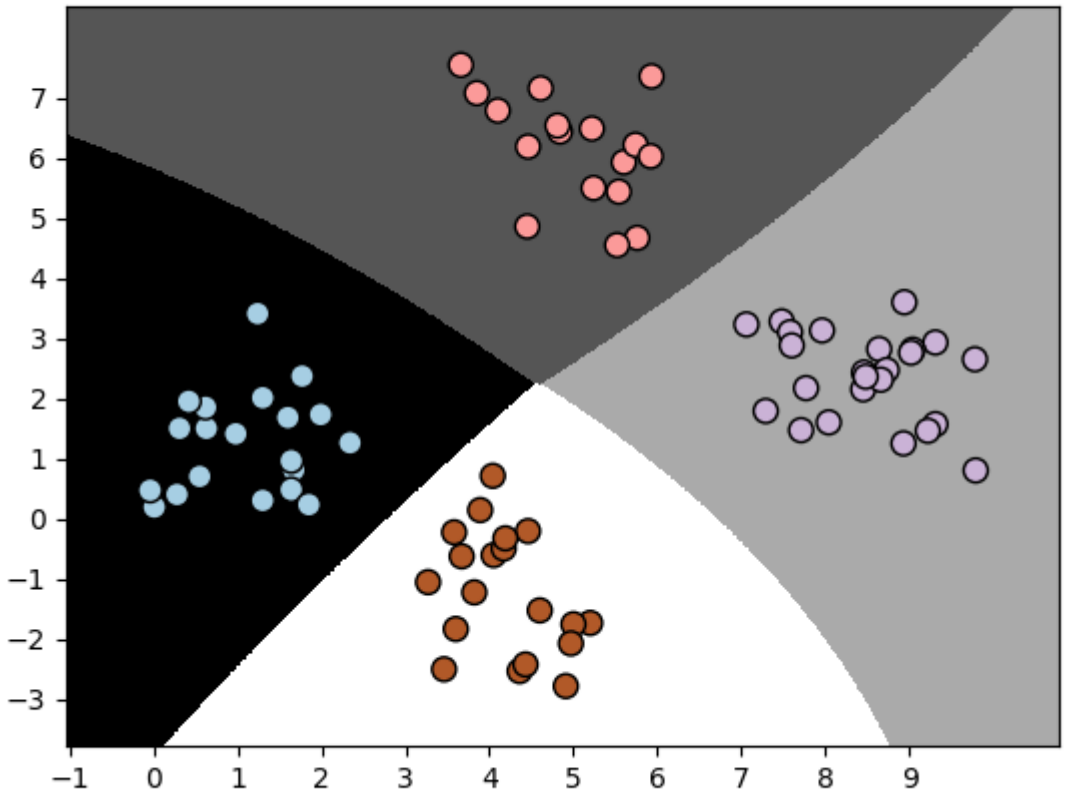

Рис.1.10 – Зображення результату класифікації тестових даних

Розділення даних дозволило більш надійно отримати відповідь, а використання функції для обчислення якості, точності та повноти дозволило більш детально вказати результат.

Завдання 5. Вивчити метрики якості класифікації

Важливими метриками якості є якість, точність, чутливість та F1. Їх обчислення відбувається завдяки порівнянню результатів з реальністю, а саме зберіганням значень TP, FN, FP, TN.

Лістинг коду файлу Task_5.py:

```
import numpy as np
import pandas as pd
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')
print(df.head())

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

conf_matr = confusion_matrix(df.actual_label.values, df.predicted_RF.values)
print("confusion_matrix:\n", conf_matr)


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))


def find_conf_matrix_values(y_true, y_pred):
    """
    :param y_true: List with true data of classification
    :param y_pred: List with predicted data of classification
    :return: 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 Chyzhmotria_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("Oleksiichuk_confusion_matrix:\n", Chyzhmotria_confusion_matrix(actual,
predicted_RF))

assert np.array_equal(Chyzhmotria_confusion_matrix(actual, predicted_RF),
                      confusion_matrix(actual, predicted_RF)), \
    'my confusion_matrix() is not correct for RF'

assert np.array_equal(Chyzhmotria_confusion_matrix(actual, predicted_LR),
                      confusion_matrix(actual, predicted_LR)), \
    'my confusion_matrix() is not correct for lR'

# Accuracy
score = accuracy_score(actual, predicted_RF)
print("Accuracy score on RF:", score)


def Chyzhmotria_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 Chyzhmotria_accuracy_score(actual, predicted_RF) == accuracy_score(actual,
predicted_RF), \
    'my accuracy_score failed RF'

assert Chyzhmotria_accuracy_score(actual, predicted_LR) == accuracy_score(actual,
predicted_LR), \
    'my accuracy_score failed LR'

print("My accuracy score on RF:", Chyzhmotria_accuracy_score(actual,
predicted_RF))
print("My accuracy score on LR:", Chyzhmotria_accuracy_score(actual,
predicted_LR))

# Recall
print('Recall score on RF:', recall_score(actual, predicted_RF))


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


assert Chyzhmotria_recal_score(actual, predicted_RF) == recall_score(actual,
predicted_RF),\
    'my recal_score fails on RF'

assert Chyzhmotria_recal_score(actual, predicted_LR) == recall_score(actual,
predicted_LR),\
    'my recal_score fails on LR'

print("My recall score on RF:", Chyzhmotria_recal_score(actual, predicted_RF))
print("My recall score on LR:", Chyzhmotria_recal_score(actual, predicted_LR))

# Precision
print("Precision score on RF:", precision_score(actual, predicted_RF))

def Chyzhmotria_precision_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
```

```
    return TP / (TP + FP)


assert Chyzhmotria_precision_score(actual, predicted_RF) ==
precision_score(actual, predicted_RF),\
    'my precision_score fails on RF'

assert Chyzhmotria_precision_score(actual, predicted_LR) ==
precision_score(actual, predicted_LR),\
    'my precision_score fails on LR'

print("My precision score on RF:", Chyzhmotria_precision_score(actual,
predicted_RF))
print("My precision score on LR:", Chyzhmotria_precision_score(actual,
predicted_LR))

# F1 score
print("F1 score on RF", f1_score(actual, predicted_RF))


def Chyzhmotria_f1_score(y_true, y_pred):
    precision = Chyzhmotria_precision_score(y_true, y_pred)
    recall = Chyzhmotria_recal_score(y_true, y_pred)
    return (2 * (precision * recall)) / (precision + recall)


assert Chyzhmotria_f1_score(actual, predicted_RF) == f1_score(actual,
predicted_RF),\
    'my f1_score fails on RF'

assert Chyzhmotria_f1_score(actual, predicted_LR) == f1_score(actual,
predicted_LR),\
    'my f1_score fails on LR'

print("My F1 score score on RF:", Chyzhmotria_f1_score(actual, predicted_RF))
print("My F1 score score on LR:", Chyzhmotria_f1_score(actual, predicted_LR))
print()


def test_thresholds(threshold: float = .5):
    print(f"Scores with threshold = {threshold}")
    predicted = (df.model_RF >= threshold).astype('int')

    print("Accuracy RF:", Chyzhmotria_accuracy_score(actual, predicted))
    print("Precision RF:", Chyzhmotria_precision_score(actual, predicted))
    print("Recall RF:", Chyzhmotria_recal_score(actual, predicted))
    print("F1 RF:", Chyzhmotria_f1_score(actual, predicted))
    print()


test_thresholds()
test_thresholds(.25)
test_thresholds(.6)
test_thresholds(.20)

# ROC
# Curve
fpr_RF, tpr_RF, thresholds_RF = roc_curve(actual, model_RF)
fpr_LR, tpr_LR, thresholds_LR = roc_curve(actual, model_LR)

# AUC
auc_RF = roc_auc_score(actual, model_RF)
auc_LR = roc_auc_score(actual, model_LR)
```

```python
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()
```

```
   actual_label  model_RF  model_LR
0             1  0.639816  0.531904
1             0  0.490993  0.414496
2             1  0.623815  0.569883
3             1  0.506616  0.443674
4             0  0.418302  0.369532
   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.1.11 – Вхідні та прогнозовані дані, перші 5 рядків

```
confusion_matrix:
 [[5519 2360]
 [2832 5047]]
Chyzhmotria_confusion_matrix:
 [[5519 2360]
 [2832 5047]]
```

Рис.1.12 – Робота власної та наданої функцій отримання матриць помилок

```
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.1.13 – Метрика моделей, отримана власними та наданими функціями

```
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 = 0.6
Accuracy RF: 0.6127681177814444
Precision RF: 0.828952239911144
Recall RF: 0.28417311841604265
F1 RF: 0.42325141776937614

Scores with threshold = 0.2
Accuracy RF: 0.5002538393197106
Precision RF: 0.5001269518852355
Recall RF: 1.0
F1 RF: 0.6667795032369992
```

Рис.1.14 – Метрика моделі RF за різних порогів

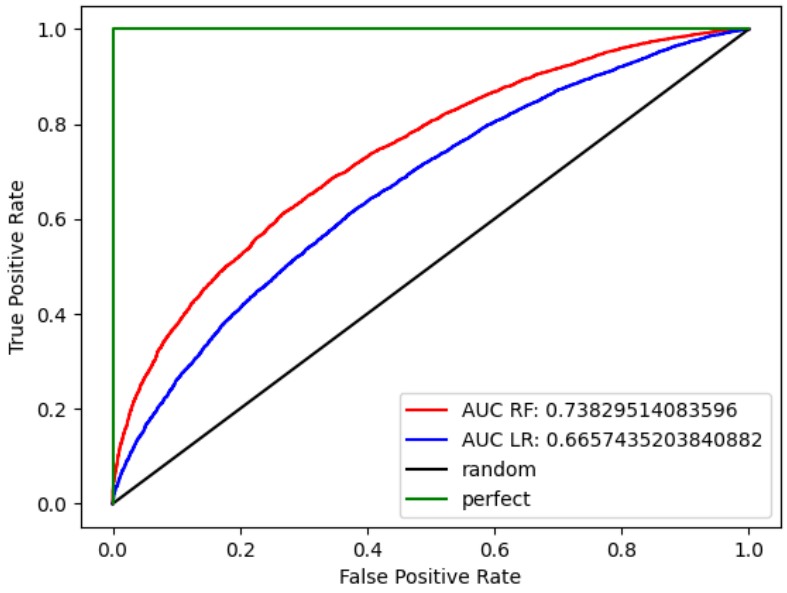

Рис.1.15 – Графік отриманих значень ROC

Завдання 6. Класифікація даних зі завдання 4 за допомоги машини опорних векторів (Support Vector Machine SVM).

Лістинг коду файлу 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)}%")
```

```
Accuracy: 100.0%
Precision: 100.0%
Recall: 100.0%
F1: 100.0%
```

Рис.1.16 – Показники класифікації з розділенням даних на 80% навчальних

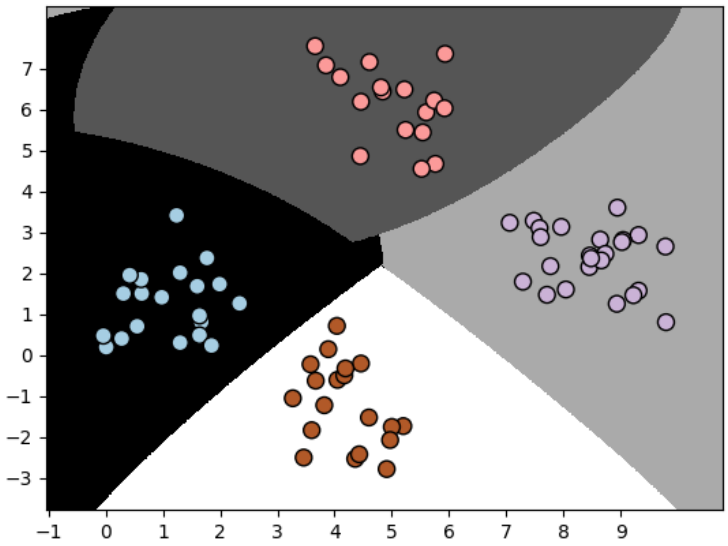

Рис.1.17 – Зображення результату класифікації тестових даних за допомоги SVM

Використання SVM надає кращі результати класифікації завдяки швидкості та простоті, проте для використання для багатокласової класифікації він не пристосований. Окрім цього, кількість даних може бути недостатньою через однакові показники.

Висновки: в ході виконання лабораторної роботи було отримано навички з: попередньої обробки даних шляхами бінаризації, виключення середнього, масштабування, нормалізації, кодування міток та закріплено на даних по варіантах; класифікації даних логістичною регресією; класифікації даних Наївним Байєсом; отримання та аналізу метрик якості класифікації; використання SVM та класифікації з використанням SVM даних. Під час аналізу метрик якості класифікації було розроблено власні функції з отримання необхідних даних та їх групування в матрицю помилок, порівняно отримані дані з даними від функцій.

Github: https://github.com/mikrorobot/Python_AI