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 2.1: 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))


# 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)
```

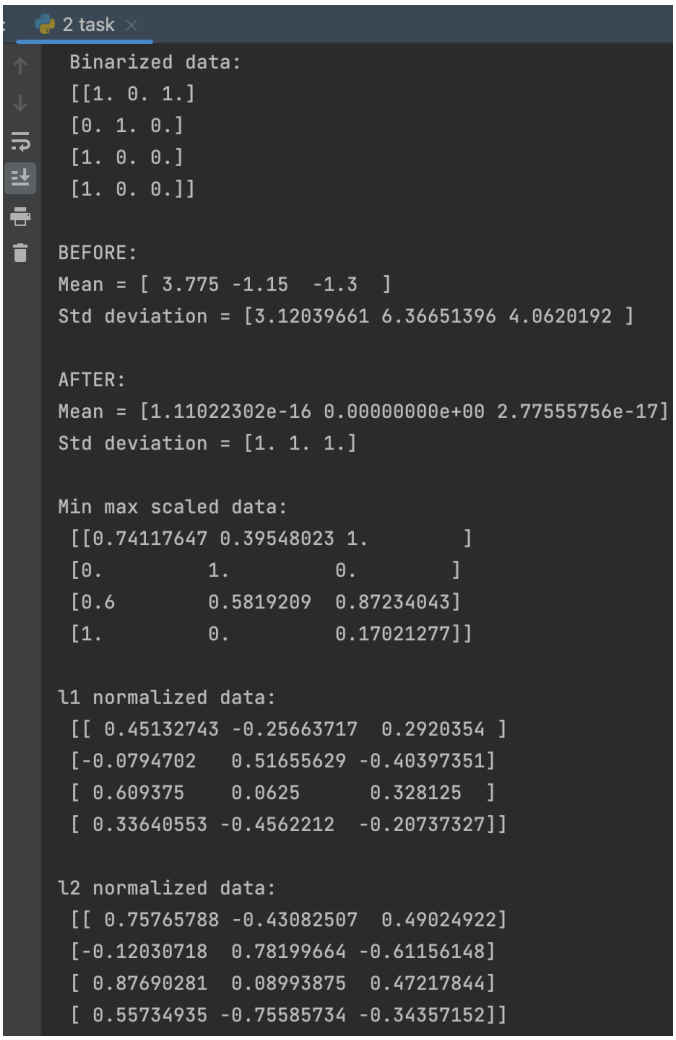


Рис. 1.1 Результат виконання завдання

Висновок: L2-нормалізація має меншу точність та надійність, ніж L1-нормалізація, адже забезпечує рівність 1 суми квадратів значень в кожному ряду, а L1 абсолютних значень. Проте L1 не дозволяє вирішувати завдання з простеженням неточності вхідних даних.

Завдання 1: Кодування міток

Лістинг програми:

```
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)
```

```python
# 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
Ыack --> 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. 1.2 Результат виконання завдання

Завдання 2: Попередня обробка нових даних

Таблиця 1

№ варіанту	Значення змінної input_data												Поріг бінаризації
14	-1.3	3.9	6.2	- 4.9	2.2	- 4.3	-2.2	6.5	4.1	-5.2	3.4	-5.2	2.2

Лістинг програми:

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


input_data = np.array([[-1.3, 3.9, 6.2],
                       [-4.9, 2.2, -4.3],
                       [-2.2, 6.5, 4.1],
                       [-5.2, 3.4, -5.2]])
```

```
# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.2).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. 1. 1.]
 [0. 0. 0.]
 [0. 1. 1.]
 [0. 1. 0.]]

BEFORE:
Mean = [-3.4  4.   0.2]
Std deviation = [1.68374582 1.57003185 5.01547605]

AFTER:
Mean = [ 1.66533454e-16 -1.38777878e-17  0.00000000e+00]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[1.         0.39534884 1.        ]
 [0.07692308 0.         0.07894737]
 [0.76923077 1.         0.81578947]
 [0.         0.27906977 0.        ]]

l1 normalized data:
 [[-0.11403509  0.34210526  0.54385965]
 [-0.42982456  0.19298246 -0.37719298]
 [-0.171875    0.5078125   0.3203125 ]
 [-0.37681159  0.24637681 -0.37681159]]

l2 normalized data:
 [[-0.17475265  0.52425796  0.83343572]
 [-0.71216718  0.31974853 -0.62496303]
 [-0.2752151   0.81313551  0.51290086]
 [-0.64182859  0.41965715 -0.64182859]]
```

Рис. 1.3 Результат виконання завдання

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)
```

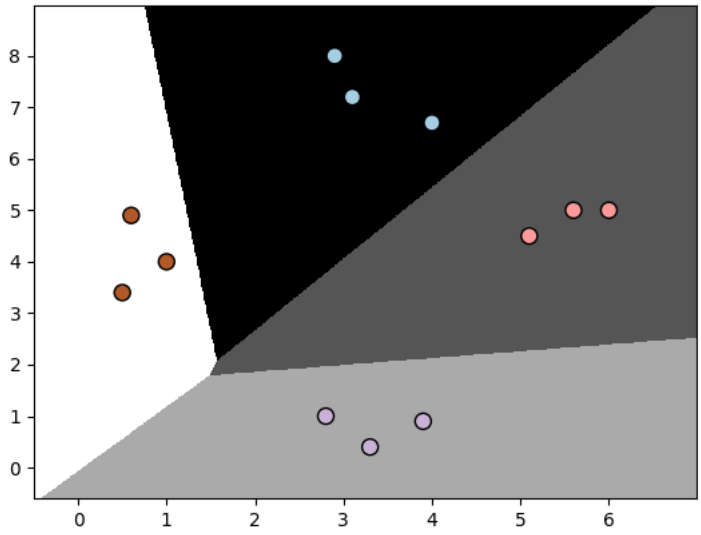


Рис. 1.4 Результат виконання завдання

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)
```

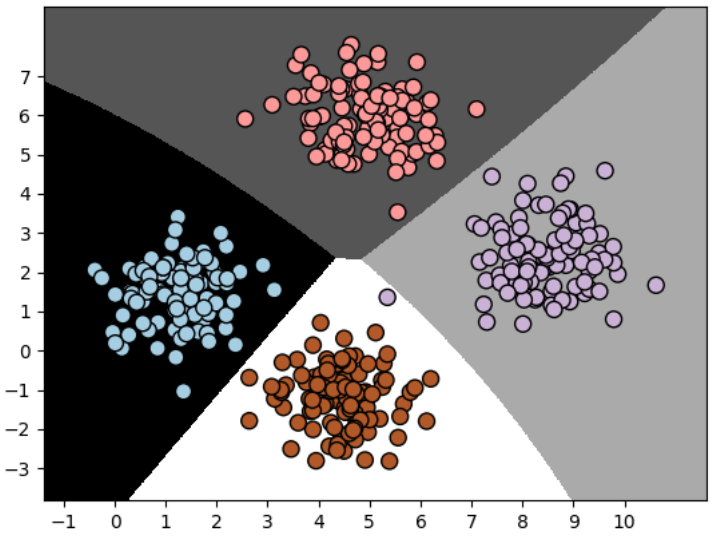


Рис. 1.5 Результат виконання без перехресної перевірки

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)


# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split.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 = train_test_split.cross_val_score(classifier, X, y,
scoring='accuracy', cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")

precision_values = train_test_split.cross_val_score(classifier, X, y,
scoring='precision_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")

recall_values = train_test_split.cross_val_score(classifier, X, y,
scoring='recall_weighted', cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")

f1_values = train_test_split.cross_val_score(classifier,X, y,
scoring='f1_weighted', cv=num_folds)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")
```

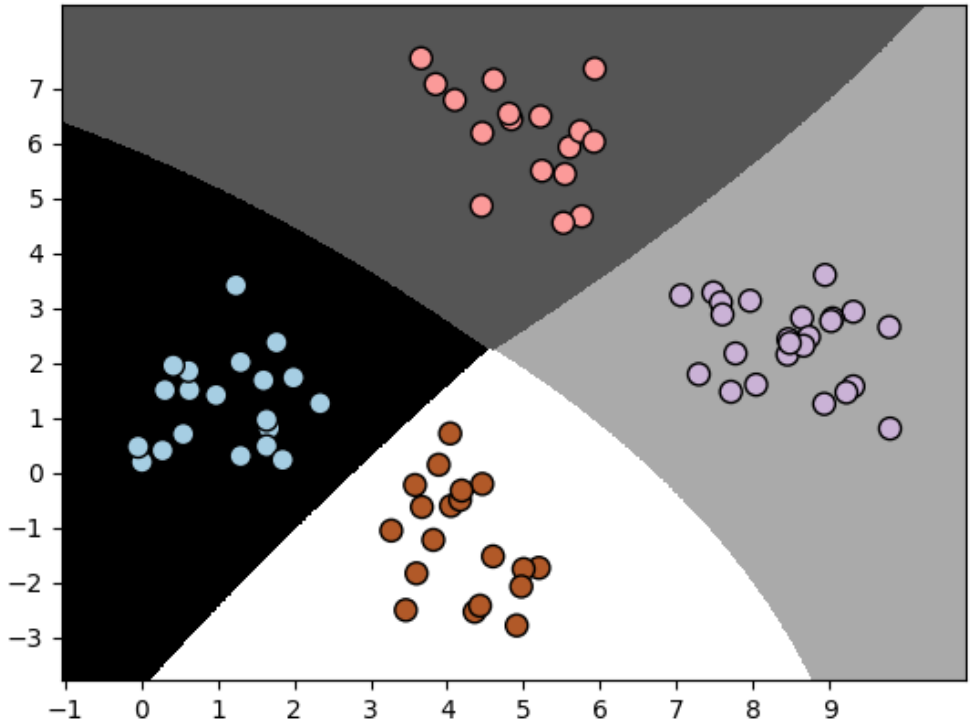


Рис. 1.6 Результат виконання завдання

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))


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 sirach_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(sirach_confusion_matrix(df.actual_label.values, df.predicted_RF.values))
print(sirach_confusion_matrix(df.actual_label.values, df.predicted_LR.values))

assert np.array_equal(sirach_confusion_matrix(df.actual_label.values,
df.predicted_RF.values),
                      confusion_matrix(df.actual_label.values,
                                       df.predicted_RF.values)),
'sirach_confusion_matrix() is not correct for RF'
assert np.array_equal(sirach_confusion_matrix(df.actual_label.values,
df.predicted_LR.values),
                      confusion_matrix(df.actual_label.values,
                                       df.predicted_LR.values)),
'sirach_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 sirach_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 sirach_accuracy_score(df.actual_label.values, df.predicted_RF.values) ==
accuracy_score(
    df.actual_label.values,
    df.predicted_RF.values), 'sirach_accuracy_score failed on assert
sirach_accuracy_score(df.actual_label.values, ' \
                         'df.predicted_LR.values) == accura-
cy_score(df.actual_label.values, df.predicted_LR.values),' \
                         'sirach_accuracy_score failed on LR'
print('Accuracy RF: % .3f' % (sirach_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Accuracy LR: % .3f' % (sirach_accuracy_score(df.actual_label.values,
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 sirach_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 sirach_recall_score(df.actual_label.values, df.predicted_RF.values) ==
recall_score(df.actual_label.values,

df.predicted_RF.values), \
    'sirach_recall_score failed on RF'
assert sirach_recall_score(df.actual_label.values, df.predicted_LR.values) ==
recall_score(df.actual_label.values,

df.predicted_LR.values), \
    'sirach_recall_score failed on LR'
print('Recall RF: %.3f' % (sirach_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall LR: %.3f' % (sirach_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 sirach_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 sirach_precision_score(df.actual_label.values, df.predicted_RF.values) ==
precision_score(
    df.actual_label.values, df.predicted_RF.values), 'sirach_precision_score
failed on RF'
assert sirach_precision_score(df.actual_label.values, df.predicted_LR.values) ==
precision_score(
    df.actual_label.values, df.predicted_LR.values), 'sirach_precision_score
failed on LR'
print('Precision RF: %.3f' % (sirach_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f' % (sirach_precision_score(df.actual_label.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 sirach_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = sirach_recall_score(y_true, y_pred)
    precision = sirach_precision_score(y_true, y_pred)
    return (2 * (precision * recall)) / (precision + recall)


assert sirach_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values,
```

```
df.predicted_RF.values), 'sirach_f1_score failed on RF'
assert sirach_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values,

df.predicted_LR.values), 'sirach_f1_score failed on LR'
print('F1 RF: %.3f' % (sirach_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 LR: %.3f' % (sirach_f1_score(df.actual_label.values,
df.predicted_LR.values)))

print('scores with threshold = 0.5')
print('Accuracy RF: %.3f' % (sirach_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f' % (sirach_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: %.3f' % (sirach_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f' % (sirach_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f' % (
    sirach_accuracy_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('Recall RF: %.3f' % (sirach_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f' % (
    sirach_precision_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('F1 RF: %.3f' % (sirach_f1_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
```

```
[[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
```

Рис. 1.7 Метрики якості класифікації

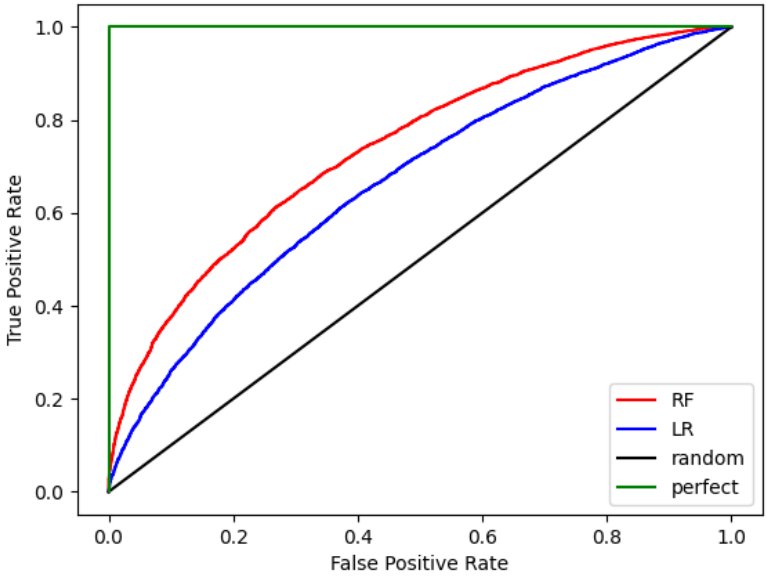


Рис. 1.8 Графік отриманих значень ROC

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.

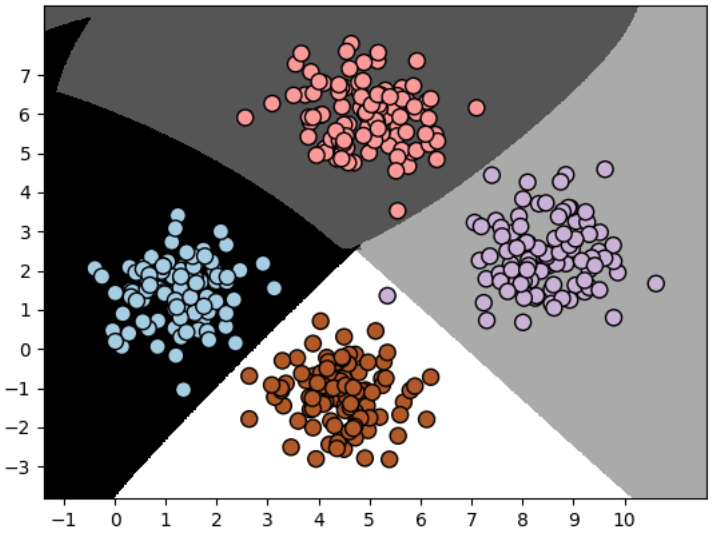
Лістинг програми:

```
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)

# # 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:", round(accuracy, 2), "%")

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

num_folds = 4
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: 100.0 %
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Рис. 1.9 Показники якості класифікатора



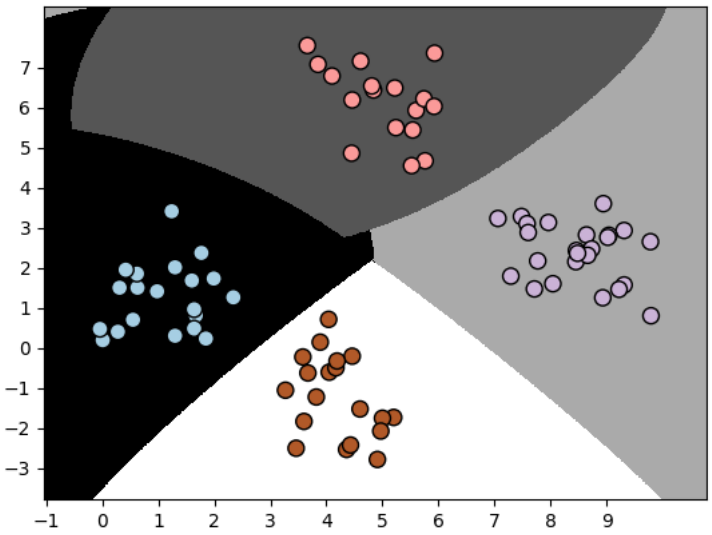


Рис. 1.11 Результат класифікації тестових даних за допомоги SVM

Висновки по використанню SVM класифікатора в порівнянні з байєсівським класифікатором: загалом отримані показники обох класифікаторів – ідентичні. Байєсівський класифікатор є менш швидким, ненадійним, але простим. Він визначає кожну ознаку як незалежну, через це важко отримати повну картину. SVM класифікатор є більш швидким, проте він підходить лише до розв'язання завдань з двома класами.

***Висновки*:** в ході виконання лабороторної роботи, було досліджено попередню обробку та класифікацію даних, використовуючи спеціальні бібліотеки та мову програмування Python.

Посилання на GitHub: https://github.com/annasirach/AI_IPZ193_Sirach