**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

**Бінарізація**

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)

Результат виконання завдання

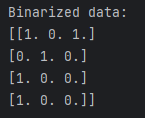


Рис. 1.1 Результат виконання завдання

**Виключення середнього**

Лістинг програми:

# 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))

Результат виконання завдання

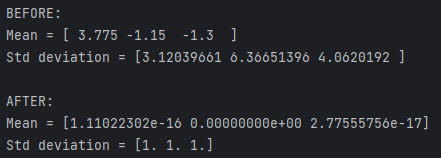


Рис. 1.2 Результат виконання завдання

**Масштабування**

Лістинг програми:

# Масштабування MinМax  
data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)  
print("\nМin max scaled data:\n", data\_scaled\_minmax)

Результат виконання завдання

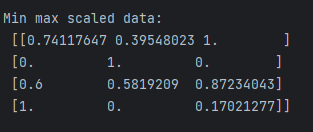


Рис. 1.3 Результат виконання завдання

**Нормалізація**

Лістинг 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)

Результат виконання завдання

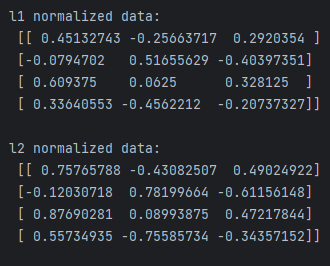


Рис. 1.4 Результат виконання завдання

**L1-нормалізація** використовує метод найменших абсолютних відхилень, що забезпечує рівність 1 суми абсолютних значень в кожному ряду, в той час як, **L2-нормалізація –** рівність 1 суми квадратів значень в кожному ряду. Тому **L1-нормалізації** вважається більш надійною по порівняно з **L2-нормалізацією**, оскільки вона менш чутлива до викидів.

**Кодування міток**

Лістинг програми:

import numpy as np  
from sklearn import preprocessing  
  
# Надання позначок вхідних даних  
input\_labels = ['red', 'Ыасk', 'red', 'green', 'Ьlack', 'yellow', 'white']  
  
# Створення кодувальника та встановлення відповідності між мітками та числа-ми  
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', 'Ыасk']  
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))

Результат виконання завдання

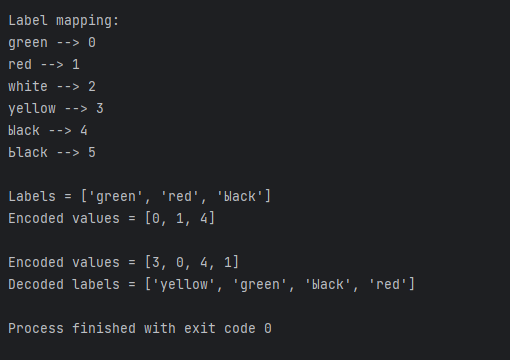


Рис. 1.5 Результат виконання завдання

**Завдання 2.2:** Попередня обробка нових даних

Таблиця 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № варіанту | Значення змінної input\_data | | | | | | | | | | | | Поріг бінаризації |
| 5 | -1.3 | 3.9 | 4.5 | -5.3 | -4.2 | -1.3 | 5.2 | -6.5 | -1.1 | -5.2 | 2.6 | -2.2 | 3.0 |

Лістинг програми:

import numpy as np  
from sklearn import preprocessing  
  
input\_data = np.array([  
 [-1.3, 3.9, 4.5],  
 [-5.3, -4.2, -1.3],  
 [5.2, -6.5, -1.1],  
 [-5.2, 2.6, -2.2]  
])  
  
# Бінаризація даних  
data\_binarized = preprocessing.Binarizer(threshold=1.8).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 MinМax  
data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)  
print("\nМin 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)

Результат виконання завдання

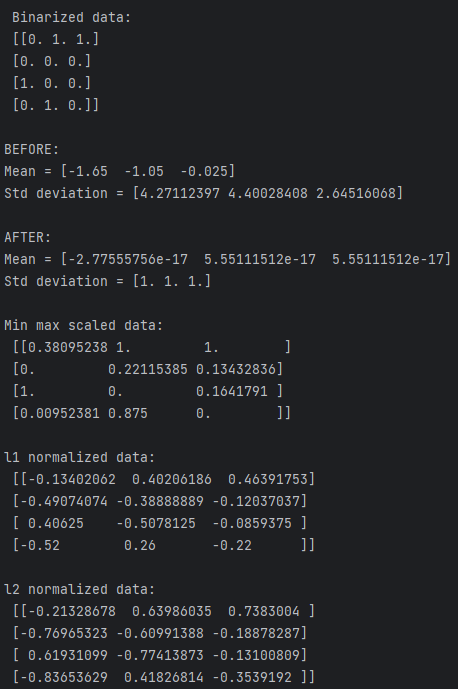


Рис. 1.6 Результат виконання завдання

**Завдання 2.3:** 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)

Результат виконання завдання

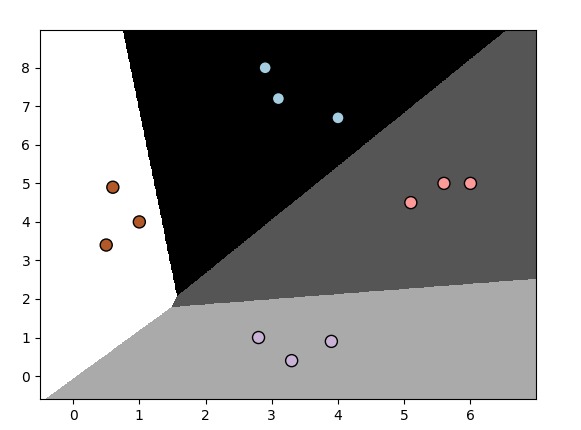


Рис. 1.7 Результат виконання завдання

**Завдання 2.4:** 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)

Результат виконання завдання

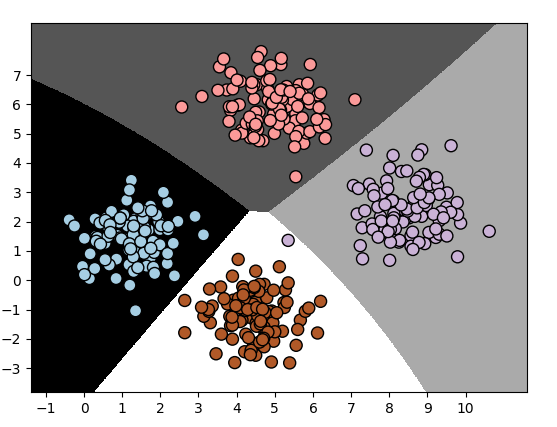


Рис. 1.8 Результат виконання завдання

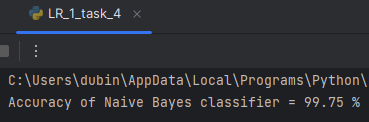


Рис. 1.9 Результат виконання завдання

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, cross\_val\_score  
from utilities import visualize\_classifier

# 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)) + "%")

Результат виконання завдання

1 прогін

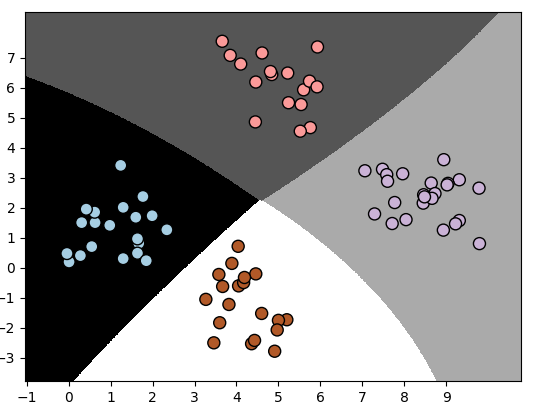


Рис. 1.10 Результат виконання завдання

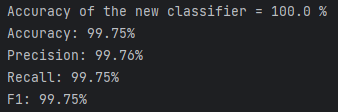


Рис. 1.11 Результат виконання завдання

2 прогін

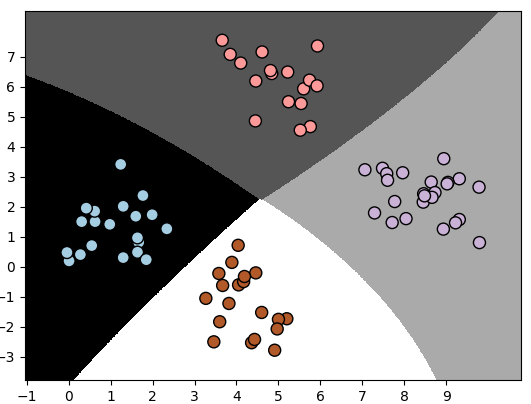


Рис. 1.12 Результат виконання завдання

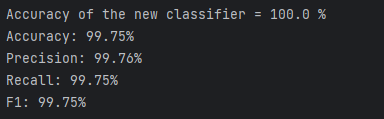


Рис. 1.13 Результат виконання завдання

Отримані результати після двох прогонів ідентичні, бо тренування відбувалися на однакових початкових значеннях.

**Завдання 2.5**: Вивчити метрики якості класифікації

Лістинг програми:

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion\_matrix  
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  
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))  
  
  
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 my\_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(my\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values))  
print(my\_confusion\_matrix(df.actual\_label.values, df.predicted\_LR.values))  
assert np.array\_equal(my\_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(my\_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'  
  
print(accuracy\_score(df.actual\_label.values, df.predicted\_RF.values))  
print(accuracy\_score(df.actual\_label.values, df.predicted\_LR.values))  
  
def my\_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 my\_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 assert my\_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' % (my\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Accuracy LR: % .3f' % (my\_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 my\_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 my\_recall\_score(df.actual\_label.values, df.predicted\_RF.values) == recall\_score(df.actual\_label.values,  
 df.predicted\_RF.values), \  
 'my\_recall\_score failed on RF'  
assert my\_recall\_score(df.actual\_label.values, df.predicted\_LR.values) == recall\_score(df.actual\_label.values,  
 df.predicted\_LR.values), \  
 'my\_recall\_score failed on LR'  
print('Recall RF: %.3f' % (my\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall LR: %.3f' % (my\_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 my\_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 my\_precision\_score(df.actual\_label.values, df.predicted\_RF.values) == precision\_score(  
 df.actual\_label.values, df.predicted\_RF.values), 'my\_precision\_score failed on RF'  
assert my\_precision\_score(df.actual\_label.values, df.predicted\_LR.values) == precision\_score(  
 df.actual\_label.values, df.predicted\_LR.values), 'my\_precision\_score failed on LR'  
print('Precision RF: %.3f' % (my\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision LR: %.3f' % (my\_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 my\_f1\_score(y\_true, y\_pred):  
 # calculates the F1 score  
 recall = my\_recall\_score(y\_true, y\_pred)  
 precision = my\_precision\_score(y\_true, y\_pred)  
 return (2 \* (precision \* recall)) / (precision + recall)  
  
  
assert my\_f1\_score(df.actual\_label.values, df.predicted\_RF.values) == f1\_score(df.actual\_label.values,  
 df.predicted\_RF.values), 'my\_f1\_score failed on RF'  
assert my\_f1\_score(df.actual\_label.values, df.predicted\_LR.values) == f1\_score(df.actual\_label.values,  
 df.predicted\_LR.values), 'my\_f1\_score failed on LR'  
print('F1 RF: %.3f' % (my\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 LR: %.3f' % (my\_f1\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
print('scores with threshold = 0.5')  
print('Accuracy RF: %.3f' % (my\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall RF: %.3f' % (my\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision RF: %.3f' % (my\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 RF: %.3f' % (my\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('')  
print('scores with threshold = 0.25')  
print('Accuracy RF: %.3f' % (  
 my\_accuracy\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('Recall RF: %.3f' % (my\_recall\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('Precision RF: %.3f' % (  
 my\_precision\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('F1 RF: %.3f' % (my\_f1\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
  
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)  
  
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()  
  
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.show()

Результат виконання завдання

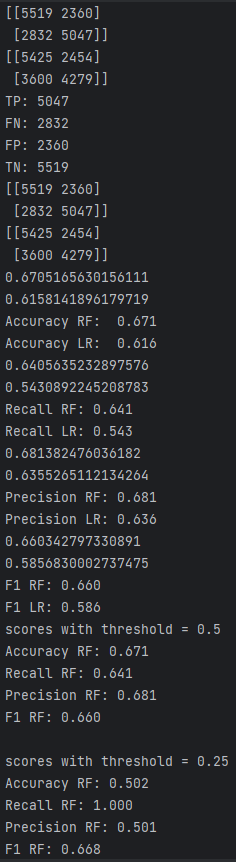


Рис. 1.14 Результат виконання завдання

Акуратність для більшого порогу є кращою за акуратність для меншого. Але чутливість навпаки для більшого порогу є меншою. Точність для порогу 0.5 виявилася більшою за точність для 0.25. Оцінка f1 є майже ідентичною для обох порогів, а так як цей показник є одним з точніших для визначення пріоритетної моделі, то можемо зробити висновок, що обидва пороги мають право життя.

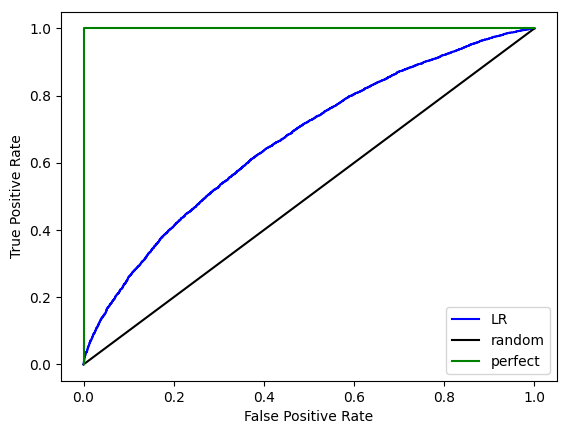


Рис. 1.15 Крива ROC для обох моделей

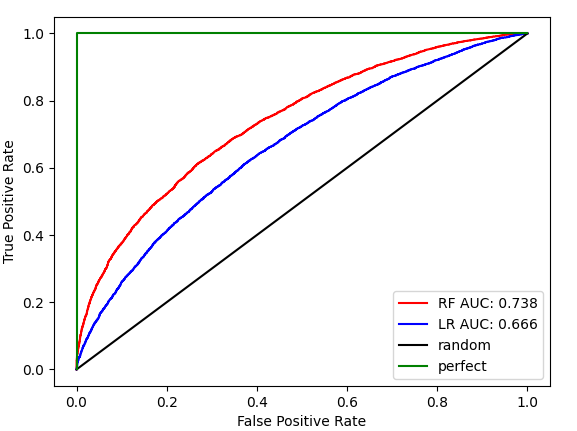


Рис. 1.16 Крива ROC для обох моделей (з урахуванням площ під кривою)

Площа під кривою для моделі RF (AUC = 0,738) краще, ніж LR (AUC = 0,666). Отже, згідно вищевказаної метрики робимо висновок, що модель RF краще.

**Завдання 2.6:** Розробіть програму класифікації даних в файлі data\_multivar\_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVМ). 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)  
  
# Обчислення якості класифікатора  
accuracy = 100.0 \* (y == y\_pred).sum() / X.shape[0]  
print("Accuracy of Support Vector Machine 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 = 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)) + "%")

Класифікація Support Vector Machine - SVМ

Результат виконання завдання

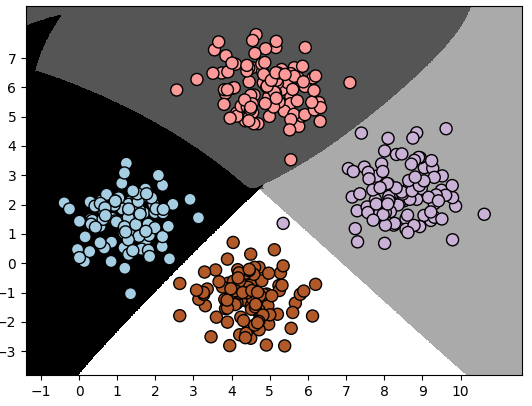


Рис. 1.17 Результат виконання завдання

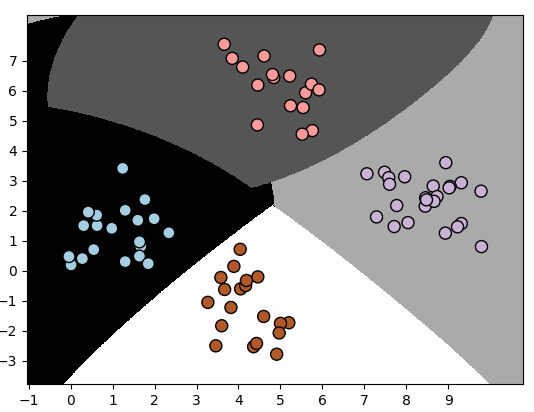


Рис. 1.18 Результат виконання завдання

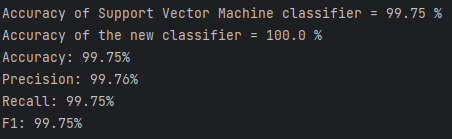


Рис. 1.19 Результат виконання завдання

Показники отримані з показниками обох класифікаторів ідентичні. Тому визначити який класифікатор краще неможливо на даному прикладі. Але зважаючи на те, що наївний байєсівський класифікатор визначає кожну ознаку як незалежну, важко отримати повну картину. Через це доцільніше використовувати класифікатор методу опорних векторів, а, також, він є найпопулярнішим методом класичної класифікації.

Посилання на GitHub: <https://github.com/BogdanStelmah/Basics-of-AI_labs>

**Висновок:** На даній лабораторній роботі мивикористовуючи спеціалізовані бібліотеки та мову програмування Python дослідили попередню обробку та класифікацію даних.