Лабораторна робота №1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

Мета роботи: 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]])
```

Завдання 2.1.1. Бінаризація.

Лістинг програми:

```
# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print("\n Binarized data:\n", data_binarized)
```

Результат виконання програми:

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

Рис. 2.1.1 – Результат виконання бінаризації.

Завдання 2.1.2. Виключення середнього.

Лістинг програми:

```
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))
```

Результат виконання програми:

```
BEFORE:
Mean = [ 3.775 -1.15  -1.3  ]
Std deviation = [3.12039661 6.36651396 4.0620192 ]

AFTER:
Mean = [1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation = [1. 1. 1.]
```

Рис. 2.1.2 – Результат виконання виключення середнього.

Завдання 2.1.3. Масштабування.

Лістинг програми:

```
# 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)
```

Результат виконання програми:

```
Min max scaled data:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]
```

Рис. 2.1.3 – Результат виконання масштабування.

Завдання 2.1.4. 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 програми:

```
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. 2.1.4 – Результат виконання нормалізації.

Чим відрізняються L1-нормалізація від L2-нормалізації?

L1-нормалізація і L2-нормалізація є двома різними методами нормалізації даних, які використовуються для приведення даних до одиничної норми (норма вектора дорівнює 1). Вони відрізняються основним способом обчислення норми та впливом на дані.

Основна відмінність полягає в обчисленні: L1-норма враховує абсолютні значення компонента, тоді як L2-норма враховує їх квадрати. Це призводить до різниці впливу на великі та малі значення. L1-нормалізація може бути більш стійкою до великих викидів (outliers) через використання абсолютних значень, тоді як L2-нормалізація може бути більш чутливою до таких викидів через квадратичний вплив.

Завдання 2.1.5. Кодування міток.

Лістинг програми:

```python
from sklearn import preprocessing

# Надання позначок вхідних даних
Input_labels = ['red', 'Ыack', 'red', 'green', 'Ыlack', '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', '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))
```

Результат виконання програми:

```
Label mapping:
green --> 0
red --> 1
white --> 2
yellow --> 3
Ыack --> 4
Ыlack --> 5

Labels = ['green', 'red', 'Ыack']
Encoded values = [0, 1, 4]

Encoded values = [3, 0, 4, 1]
Decoded labels = ['yellow', 'green', 'Ыack', 'red']
```

Рис. 2.1.5 – Результат виконання кодування міток.

Завдання 2.2. Попередня обробка нових даних.

№ варіанту	Значення змінної	Поріг бінаризації
6	2.3, -1.6, 6.1, -2.4, -1.2, 4.3, 3.2, 5.5, -6.1, -4.4, 1.4, -1.2	2.1

Лістинг програми:

```
from sklearn import preprocessing

# Надання позначок вхідних даних
Input_labels = [2.3, -1.6, 6.1, -2.4, -1.2, 4.3, 3.2, 5.5, -6.1, -4.4, 1.4, -1.2]

# Бінарізація
binarizer = preprocessing.Binarizer(threshold=2.1)
input_data_binarized = binarizer.transform([Input_labels])
print("Binarized data:\n", input_data_binarized)

# Виключення середнього
input_data_mean_removed = preprocessing.scale(Input_labels)
print("\nMean removed data:\n", input_data_mean_removed)

# Масштабування
min_max_scaler = preprocessing.MinMaxScaler()
input_data_scaled = min_max_scaler.fit_transform([Input_labels])
print("\nScaled data:\n", input_data_scaled)

# Нормалізація
input_data_normalized_l1 = preprocessing.normalize([Input_labels], norm='l1')
input_data_normalized_l2 = preprocessing.normalize([Input_labels], norm='l2')
print("\nNormalized data l1:\n", input_data_normalized_l1)
print("\nNormalized data l2:\n", input_data_normalized_l2)
```

Результат виконання:

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

Mean removed data:
 [ 0.48285332 -0.55850776  1.49751285 -0.77212029 -0.4517015   1.01688465
  0.72316742  1.33730345 -1.76007825 -1.30615162  0.24253923 -0.4517015 ]

Scaled data:
 [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

Normalized data l1:
 [[ 0.05793451 -0.04030227  0.15365239 -0.0604534  -0.0302267   0.10831234
   0.08060453  0.13853904 -0.15365239 -0.11083123  0.03526448 -0.0302267 ]]

Normalized data l2:
 [[ 0.1757775  -0.12228     0.46619249 -0.18342    -0.09171     0.32862749
   0.24455999  0.42033749 -0.46619249 -0.33626999  0.106995   -0.09171   ]]
```

Рис. 2.2.1 – Результат виконання попередньої обробки нових даних.

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор.

Лістинг програми:

```
import numpy as np
from sklearn import linear_model
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:

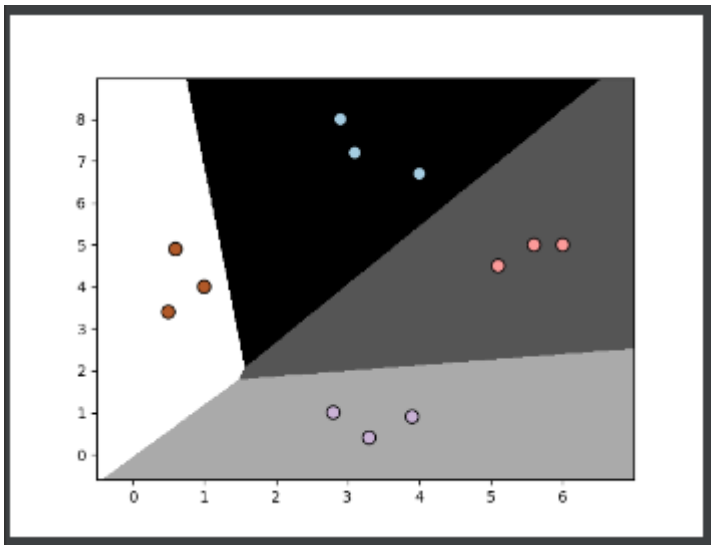

Рис. 2.3.1 – Результат виконання класифікації логістичною регресією або логістичного класифікатора.

Завдання 2.4. Класифікація наївним байєсовським класифікатором.

Лістинг програми:

```
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

# 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)) + "%")

# Новий прогін
# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X_test, y_test, 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)
```

Результат виконання:

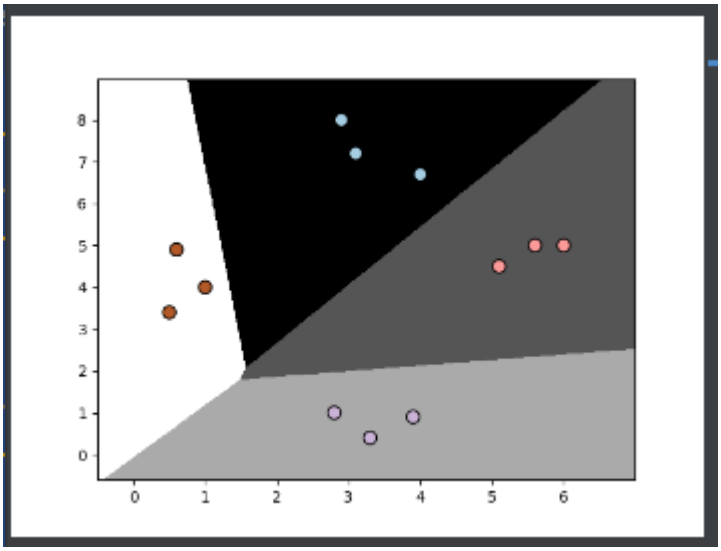

Рис. 2.4.1 – Результат виконання класифікації наївним байєсовським класифікатором (1).

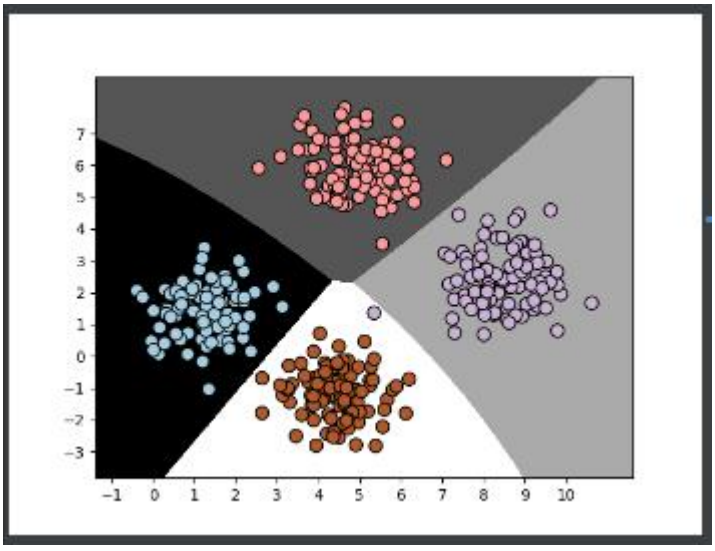

Рис. 2.4.2 – Результат виконання класифікації наївним байєсовським класифікатором (2).

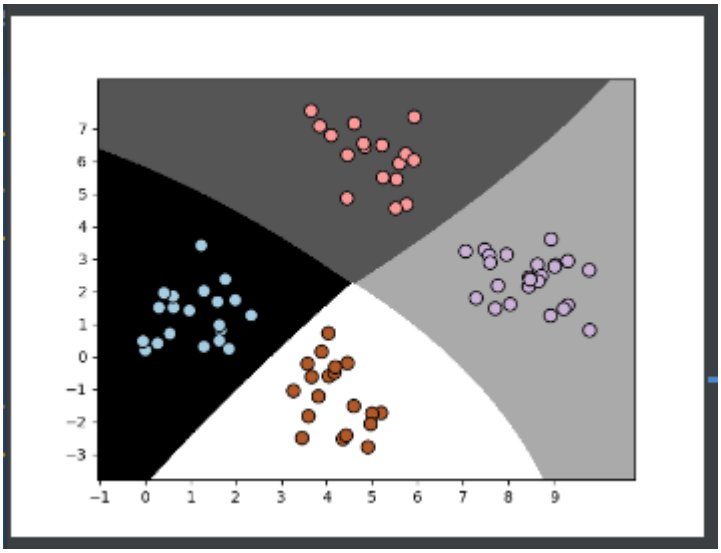

Рис. 2.4.3 – Результат виконання класифікації наївним байєсовським класифікатором (3).

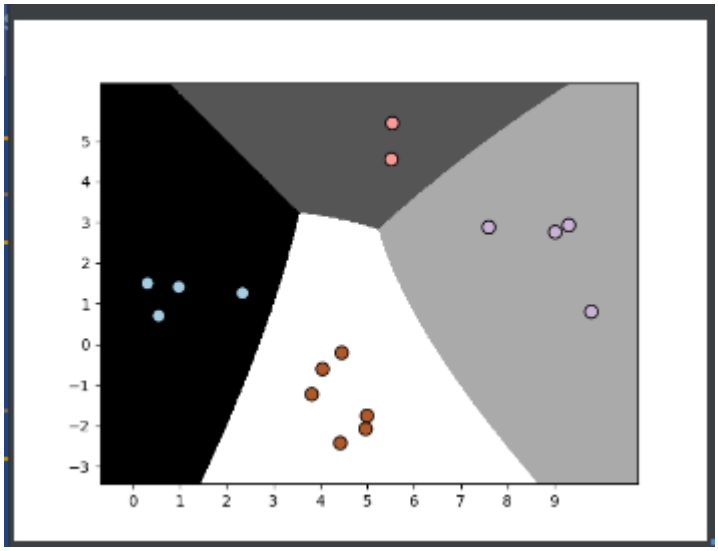

Рис. 2.4.4 – Результат виконання класифікації наївним байєсовським класифікатором (4).

```
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%
Accuracy of the new classifier = 100.0 %
```

Рис. 2.4.5 – Результат виконання класифікації наївним байєсовським класифікатором (5).

Висновок: Висновок полягає в тому, що розділення даних на тренувальні і тестові набори, а також використання крос-валідації допомагає уникнути перенавчання та дозволяє отримати більш надійні оцінки якості класифікатора. У даному випадку, класифікатор наївного байєса виявився придатним для задачі класифікації з прийнятною точністю на нових даних.

Завдання 2.5. Вивчити метрики якості класифікації.

Лістинг програми:

```
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix, roc_curve
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_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 >= 0.5).astype('int')
df['predicted_LR'] = (df.model_LR >= 0.5).astype('int')
df.head()


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 grunytsky_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]])


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

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


def grunytsky_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 grunytsky_accuracy_score(df.actual_label.values, df.predicted_RF.values) ==
accuracy_score(
    df.actual_label.values, df.predicted_RF.values), 'grunytsky_accuracy_score
failed on RF'
assert grunytsky_accuracy_score(df.actual_label.values, df.predicted_LR.values) ==
accuracy_score(
    df.actual_label.values, df.predicted_LR.values), 'grunytsky_accuracy_score
failed on LR'
print('Accuracy RF: %.3f' % (grunytsky_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Accuracy LR: %.3f' % (grunytsky_accuracy_score(df.actual_label.values,
df.predicted_LR.values)))


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


assert grunytsky_recall_score(df.actual_label.values, df.predicted_RF.values) ==
recall_score(df.actual_label.values,

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

df.predicted_LR.values), 'grunytsky_recall_score failed on LR'
print('Recall RF: %.3f' % (grunytsky_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall LR: %.3f' % (grunytsky_recall_score(df.actual_label.values,
```

```
df.predicted_LR.values)))


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


assert grunytsky_precision_score(df.actual_label.values, df.predicted_RF.values)
== precision_score(
    df.actual_label.values, df.predicted_RF.values), 'grunytsky_precision_score
failed on RF'
assert grunytsky_precision_score(df.actual_label.values, df.predicted_LR.values)
== precision_score(
    df.actual_label.values, df.predicted_LR.values), 'grunytsky_precision_score
failed on LR'

print('Precision RF: %.3f' % (grunytsky_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f' % (grunytsky_precision_score(df.actual_label.values,
df.predicted_LR.values)))


def grunytsky_f1_score(y_true, y_pred):
    recall = grunytsky_recall_score(y_true, y_pred)
    precision = grunytsky_precision_score(y_true, y_pred)
    return (2 * (precision * recall)) / (precision + recall)


assert grunytsky_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values,

df.predicted_RF.values), 'grunytsky_f1_score failed on RF'
assert grunytsky_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values,

df.predicted_LR.values), 'grunytsky_f1_score failed on LR'
print('F1 RF: %.3f' % (grunytsky_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 LR: %.3f' % (grunytsky_f1_score(df.actual_label.values,
df.predicted_LR.values)))

print('scores with threshold = 0.5')
print('Accuracy RF: %.3f' % (grunytsky_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f' % (grunytsky_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: %.3f' % (grunytsky_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f' % (grunytsky_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f' % (
    grunytsky_accuracy_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('Recall RF: %.3f' % (grunytsky_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f' % (
    grunytsky_precision_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('F1 RF: %.3f' % (grunytsky_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_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.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:

```
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
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
```

Рис. 2.5.1 – Результат виконання завдання (1).

```
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. 2.5.2 – Результат виконання завдання (2).

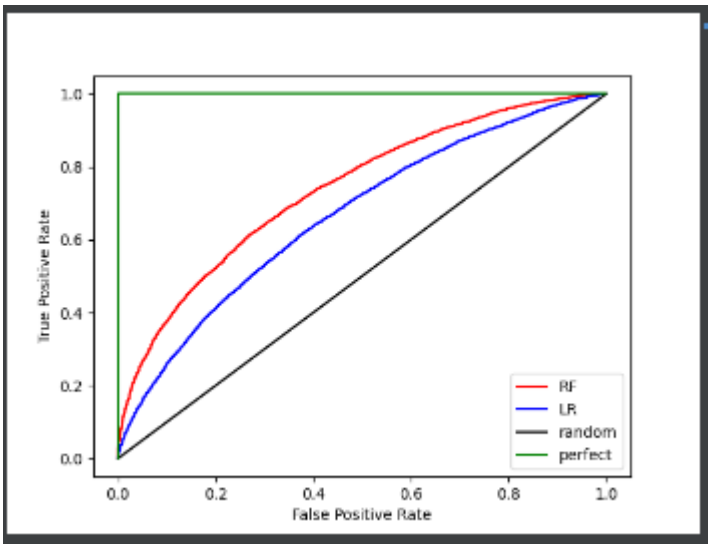

Рис. 2.5.3 – Крива ROC (1).

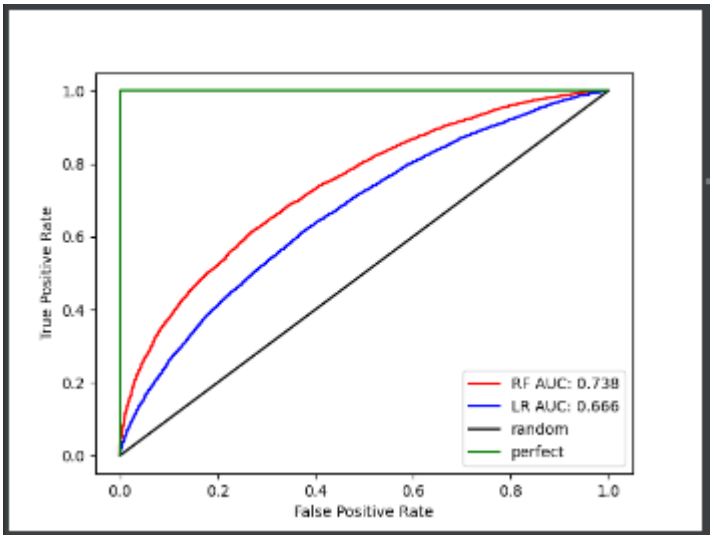

Рис. 2.5.4 – Крива ROC (2).

Порівнявши результати для різних порогів, були зроблені такі висновки:

- Вибір порогу відсічення має велике значення для балансу між точністю та чутливістю моделі.
- Поріг 0.5 є типовим і показує прийнятну якість класифікації для обох моделей.

Яка з двох моделей краща (RF та LR)?

- Оцінюючи яку з двох моделей, Random Forest (RF) або Logistic Regression (LR), краще використовувати, потрібно враховувати конкретну задачу та вимоги до моделі.

Завдання 2.6. 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 pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
from sklearn.naive_bayes import GaussianNB

data = pd.read_csv('data_multivar_nb.txt', header=None)

X = data.iloc[:, :-1]
y = data.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

svm_model = SVC(kernel='linear')

svm_model.fit(X_train, y_train)

y_pred = svm_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
confusion = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nPrecision:", precision)
print("\nRecall:", recall)
print("\nF1 score:", f1)
```

```
print("\nConfusion Matrix:\n", confusion)

nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

x_pred = nb_classifier.predict(X_test)

nb_accuracy = accuracy_score(y_test, x_pred)
nb_precision = precision_score(y_test, x_pred, average='weighted')
nb_recall = recall_score(y_test, x_pred, average='weighted')
nb_f1 = f1_score(y_test, x_pred, average='weighted')
nb_confusion = confusion_matrix(y_test, x_pred)

print("\nIndicators of the naive Bayesian classifier")
print("\nAccuracy:", nb_accuracy)
print("\nPrecision:", nb_precision)
print("\nRecall:", nb_recall)
print("\nF1 score:", nb_f1)
print("\nConfusion Matrix:\n", nb_confusion)
```

Результат виконання:

```
Accuracy: 0.9875

Precision: 0.9885416666666667

Recall: 0.9875

F1 score: 0.9876263902932255

Confusion Matrix:
 [[22  0  0  0]
 [ 0 25  0  0]
 [ 0  0 21  1]
 [ 0  0  0 11]]
```

Рис. 2.6.1 – Результат виконання завдання (1).

```
Indicators of the naive Bayesian classifier

Accuracy: 0.9875

Precision: 0.9885416666666667

Recall: 0.9875

F1 score: 0.9876263902932255

Confusion Matrix:
 [[22  0  0  0]
 [ 0 25  0  0]
 [ 0  0 21  1]
 [ 0  0  0 11]]
```

Рис. 2.6.2 – Результат виконання завдання (2).

Висновок: Модель Support Vector Machine (SVM) виявилася кращою для цієї задачі класифікації порівняно з наївним байєсовським класифікатором. SVM має більш високу точність та здатність до класифікації, що робить його більш підходящим для даного завдання.

Посилання на репозиторій: https://github.com/GrunytskyDmytro/Lab1_AI

Висновок по лабораторній роботі: в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив попередню обробку та класифікацію даних.