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

GitHub репозиторій: https://github.com/VovaYanko/SAI

Завдання 1

```
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]])

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

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))

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:
 [[1. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]

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.]

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. 1.1. Нормалізація даних

L1 нормалізація є більш надійною у порівняні з L2, бо вона базується на сумі абсолютних значень, яка є менш чутливою до викидів, ніж сума квадратів, яка є чустливою до аномалій.

```
import numpy as np
from sklearn import preprocessing

input_labels = ['red', 'black', 'red', 'green', 'black', 'yellow', 'white']

encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)

print("Label mapping:")
for i, item in enumerate(encoder.classes_):
  print(item, '-->', i)

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

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:
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. 1.2. Кодування міток

З результату виконання можна побачити, що кожній мітці присвоюється певне цифрове значення і відовідно до встановленної відповідності відбувається кодування та декодування.

Завдання 2

№ варіанту	Значення змінної input_data												Поріг бінаризації
22.	-1.6	3.9	4.5	-4.3	4.2	3.3	-5.2	-6.5	5.1	-5.2	2.6	-2.2	3.8

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

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]])

data_binarized = preprocessing.Binarizer(threshold=3.8).transform(input_data)
print("\n Binarized data:\n", data_binarized)

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))

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. 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.]

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. 1.3. Результат виконання бінаризації, виключення середнього, масштабування та нормалізації

Завдання 3

```
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)
```

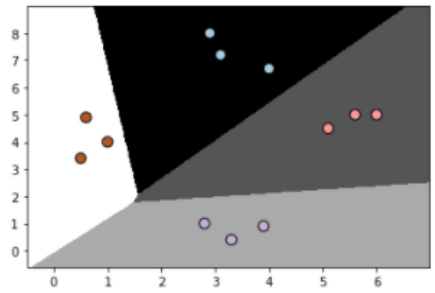


Рис. 1.4. Візуалізація класифікації логістичною регресією

Завдання 4

Результат класифікації наївним байєсовським класифікатором та вимірювання його якості наведені на рис. 1.5.

```
[3] 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

    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(accuracy)
    print("Accuracy of Naive Bayes classifier =", round(accuracy,
    2), "%")
    visualize_classifier(classifier, X, y)
```

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

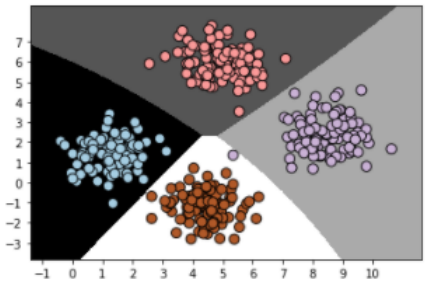
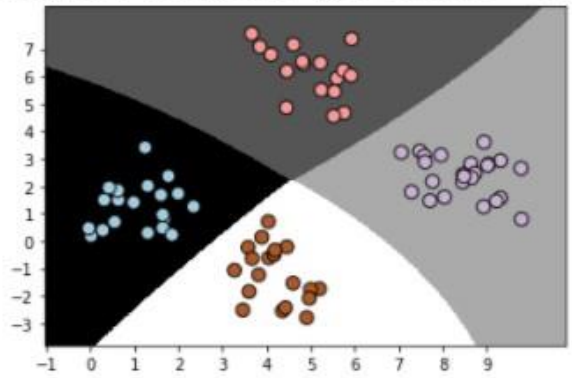


Рис. 1.5. Класифікація наївним байєсовським класифікатором

Під час першого та другого прогонів random_state залишався незмінним, що призводило до генерації однакових наборів даних для навчання та тестування класифікатора й результат залишився незмінним.

```
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

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 = GaussianNB()
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)

accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print(accuracy)
print("Accuracy of the new Naive Bayes classifier =", round(accuracy, 2), "%")
visualize_classifier(classifier, 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)) + "%")
```

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




```
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Рис. 1.6. Класифікація наївним байєсовським класифікатором з обчисленням якості, точності та повноти

Завдання 5

Відповідно до результатів порівняння моделей, можна сказати, що модель RF, краще себе показала при порозі 0.5, а модель LR при 0.25. Проте, відповідно до результатів при кроці 0.5, якісь та точніш значно вищі, що свідчить що все ж таки модель RF краща при більш оптимальному кроці, але остаточний результат повинен базуватися на порівнянні різних кроків.

```
print('\nscores with threshold = 0.5')
print('Accuracy RF: %.3f'%(sitailo_accuracy_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall RF: %.3f'%(sitailo_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision RF: %.3f'%(sitailo_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 RF: %.3f'%(sitailo_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('Accuracy LR: %.3f'%(sitailo_accuracy_score(df.actual_label.values, df.predicted_LR.values)))
print('Recall LR: %.3f'%(sitailo_recall_score(df.actual_label.values, df.predicted_LR.values)))
print('Precision LR: %.3f'%(sitailo_precision_score(df.actual_label.values, df.predicted_LR.values)))
print('F1 LR: %.3f'%(sitailo_f1_score(df.actual_label.values, df.predicted_LR.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f'%(sitailo_accuracy_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f'%(sitailo_recall_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f'%(sitailo_precision_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('F1 RF: %.3f'%(sitailo_f1_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Accuracy LR: %.3f'%(sitailo_accuracy_score(df.actual_label.values, (df.model_LR >= 0.25).astype('int').values)))
print('Recall LR: %.3f'%(sitailo_recall_score(df.actual_label.values, (df.model_LR >= 0.25).astype('int').values)))
print('Precision LR: %.3f'%(sitailo_precision_score(df.actual_label.values, (df.model_LR >= 0.25).astype('int').values)))
print('F1 LR: %.3f'%(sitailo_f1_score(df.actual_label.values, (df.model_LR >= 0.25).astype('int').values)))
```

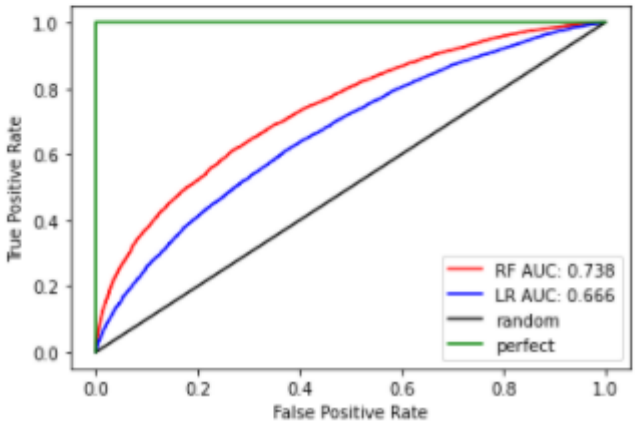
```
TP: 5047
FN: 2832
FP: 2360
TN: 5519
Accuracy RF: 0.671
Accuracy LR: 0.616
Recall RF: 0.641
Recall LR: 0.543
Precision RF: 0.681
Precision LR: 0.636
F1 RF: 0.660
F1 LR: 0.586

scores with threshold = 0.5
Accuracy RF: 0.671
Recall RF: 0.641
Precision RF: 0.681
F1 RF: 0.660
Accuracy LR: 0.616
Recall LR: 0.543
Precision LR: 0.636
F1 LR: 0.586

scores with threshold = 0.25
Accuracy RF: 0.502
Recall RF: 1.000
Precision RF: 0.501
F1 RF: 0.668
Accuracy LR: 0.503
Recall LR: 0.999
Precision LR: 0.501
F1 LR: 0.668
```

Рис. 1.7. Порівння моделей RF та LF на кроках 0.25 та 0.5

Порівнявши криві ROC, там площу під ними, можна зробити висновок що модель RF є кращою за її характеристиками на різних кроках (рис. 1.8).

```
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)

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()
```

```
AUC RF:0.738
AUC LR:0.666
```

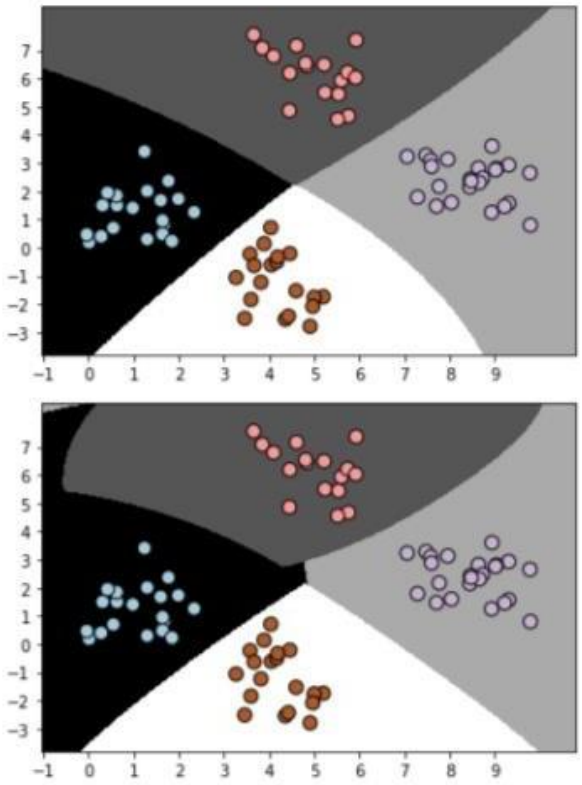


1.8. Порівняння моделей за допомогою кривих ROC

```python
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
from sklearn import preprocessing
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()
classifier_svm.fit(X_train, y_train)

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

y_pred_svm = classifier_svm.predict(X_test)
y_pred_nb = classifier_nb.predict(X_test)

num_folds = 3

print('Naive Bayes:')
accuracy_values = cross_val_score(classifier_nb, X, y, scoring = 'accuracy', cv = num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2))
+ "%")

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

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

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

print('\nSVM:')
accuracy_values = cross_val_score(classifier_svm, X, y, scoring = 'accuracy', cv = num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2))
+ "%")

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

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

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

visualize_classifier(classifier_nb, X_test, y_test)
visualize_classifier(classifier_svm, X_test, y_test)
```

1.9. Код програми для порівняння класифікаторів наївного байєса та SVM

```
Naive Bayes:
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%

SVM:
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```



1.10. Результат порівняння класифікаторів наївного байєса та SVM

Відповідно до отриманих результатів, можно зробити висновок, що обидва класифікатори дають гарний результат, але на основі візуалізації можна зробити висновок, що SVM є більш гнучким, бо межа заснована на розподілі між різними класами. Це спричинено тим, що наївний байєес відкидає можливість поєднання декількох характеристик, а SVM припускає таку можливість.

Висновки: на даній лабораторній роботі я дослідив попередню обробку та класифікацію даних використовуючи спеціалізовані бібліотеки та мову програмування Python. Також, я дізнався про класифікатори та навчився їх порівнювати.