Лабораторна робота №1 (Варіант 27)

Лістинг:

```
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],
    [5.1, -2.9, 3.3]
                       ])


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

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

# Виключення 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)
```

Результат виконання:

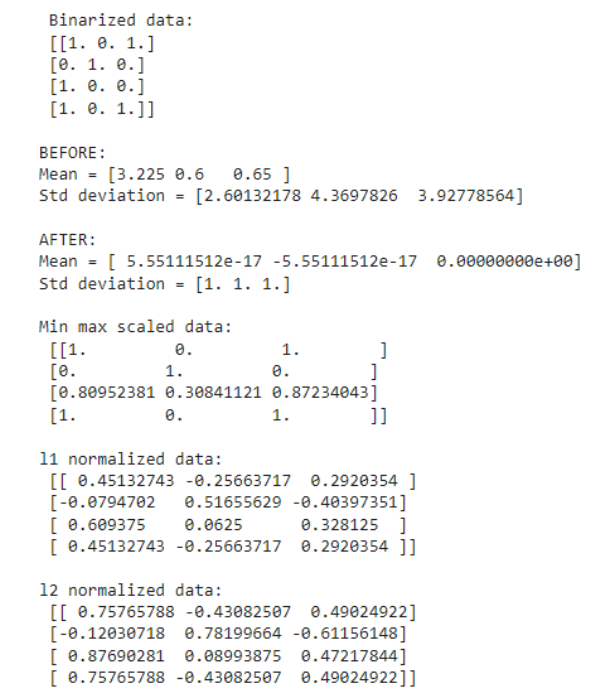


Рис. 1.

L1 нормвлізація та L2 нормалізація відрізняється тим що: L1 нормалізація підходить для масивів значень який мають дуже великі шанси викидів. L1 менш чутлива до викидів, що може бути корисним у випадках з "шумними" даними, L1 часто веде до того, що деякі значення стають нульовими. Це корисно, коли вам потрібно спростити модель, усунувши непотрібні параметри; L2 більш чутлива до викидів, оскільки їх вплив на квадратну норму більший але через це вона не робить жоден з компонентів нульовим, що дозволяє зберігти всі ознаки.

L1-нормалізація: Коли ви хочете спростити модель або у вас багато непотрібних параметрів.

L2-нормалізація: Коли вам важливо зберегти всі ознаки, і ви хочете зменшити вплив великих значень.

L1 нормалізація вважається більш надійною.

Лістинг:

```
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)

# Виведення відображення
print("\nLabel mapping:")
for i, item in enumerate(encoder.classes_) :
  print(item, '-->', i)

# перетворення міток за допомогою кодувальника
test_labels = ['green', 'red', 'black']
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:

```
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. 2

Завдання 2

У коді програми попереднього завдання поміняйте дані по рядках (значення змінної input_data) на значення відповідно варіанту таблиці 1 та виконайте операції: Бінарізації, Виключення середнього, Масштабування, Нормалізації.

Варіант обирається відповідно номера за списком групи відповідно до таблиці 1.

27.	4.6	3.9	-3.5	-2.9	4.1	3.3	2.2	8.8	-4.1	3.9	2.4	4.2	2.2

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([
    [4.6, 3.9, -3.5],
    [-2.9, 4.1, 3.3],
    [2.2, 8.8, -4.1],
    [3.9, 2.4, 4.2]
                       ])


data_binarized =
preprocessing.Binarizer(threshold=2.2).transform(input_data)
print("\n Binarized data:\n", data_binarized)

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

# Виключення 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)
```

Результат виконання:

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

BEFORE:
Mean = [ 1.95   4.8   -0.025]
Std deviation = [2.93300188 2.40104144 3.79432141]

AFTER:
Mean = [-2.77555756e-17  1.11022302e-16  5.55111512e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[1.         0.234375   0.07228916]
 [0.         0.265625   0.89156627]
 [0.68       1.         0.        ]
 [0.90666667 0.         1.        ]]

l1 normalized data:
 [[ 0.38333333  0.325      -0.29166667]
 [-0.2815534   0.39805825  0.32038835]
 [ 0.14569536  0.58278146 -0.27152318]
 [ 0.37142857  0.22857143  0.4       ]]

l2 normalized data:
 [[ 0.65970588  0.55931585 -0.50195013]
 [-0.4825966   0.68229174  0.54916164]
 [ 0.22100788  0.88403154 -0.41187833]
 [ 0.62764591  0.38624364  0.67592637]]
```

Рис. 3

Лістинг:

```
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:

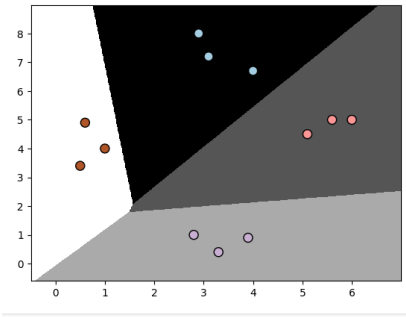


Рис. 4.

Завдання 2.4. Класифікація наївним байєсовським класифікатором

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

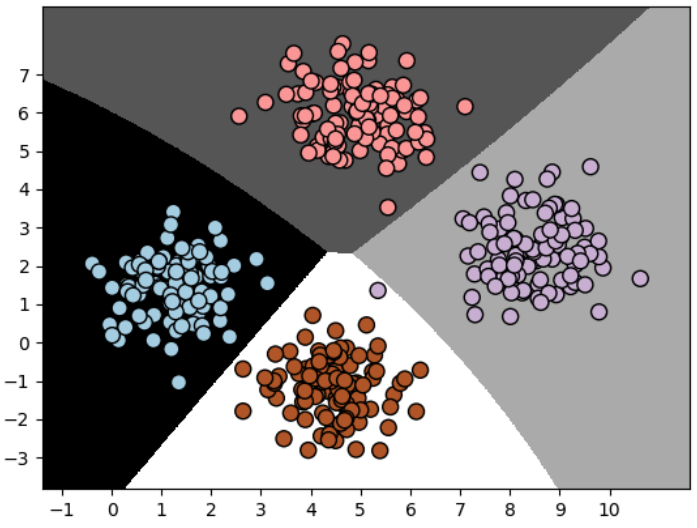


Рис. 5

Лістинг:

```
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


# 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)


# Крос-валідація

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:

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

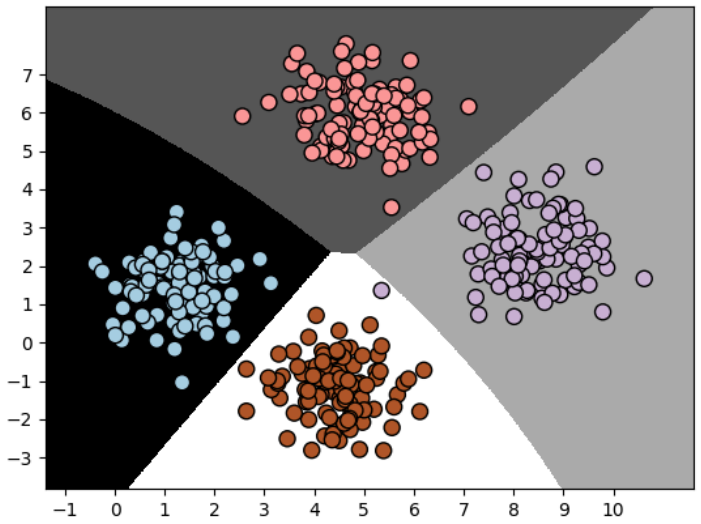
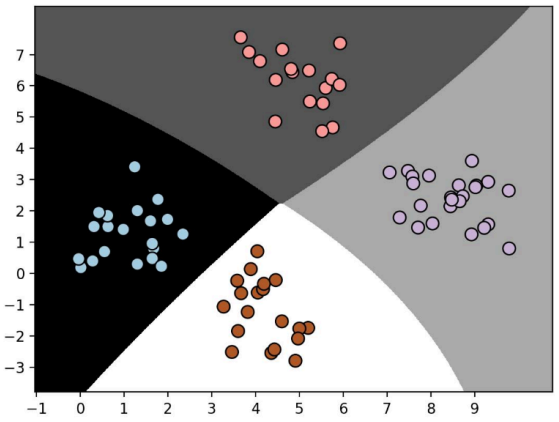


Рис. 6

Результат виконання:




```
Accuracy of the new classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Рис. 7

Після перехресної перевірки, наївний байєсовський класифікатор демонструє більш реалістичну якість класифікації. Використання окремих тренувальних і тестових наборів даних дозволило уникнути помилок, пов'язаних із використанням тих самих даних для навчання і тестування.
Як видно на скрінах другий метод більш точно розподілив точки (На першому рисунку одна з них виходить за діапазон), з чого можна зробити висновок що використання перехресної перевірки є більш надійним.

Завдання 2.5. Вивчити метрики якості класифікації

```
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
```

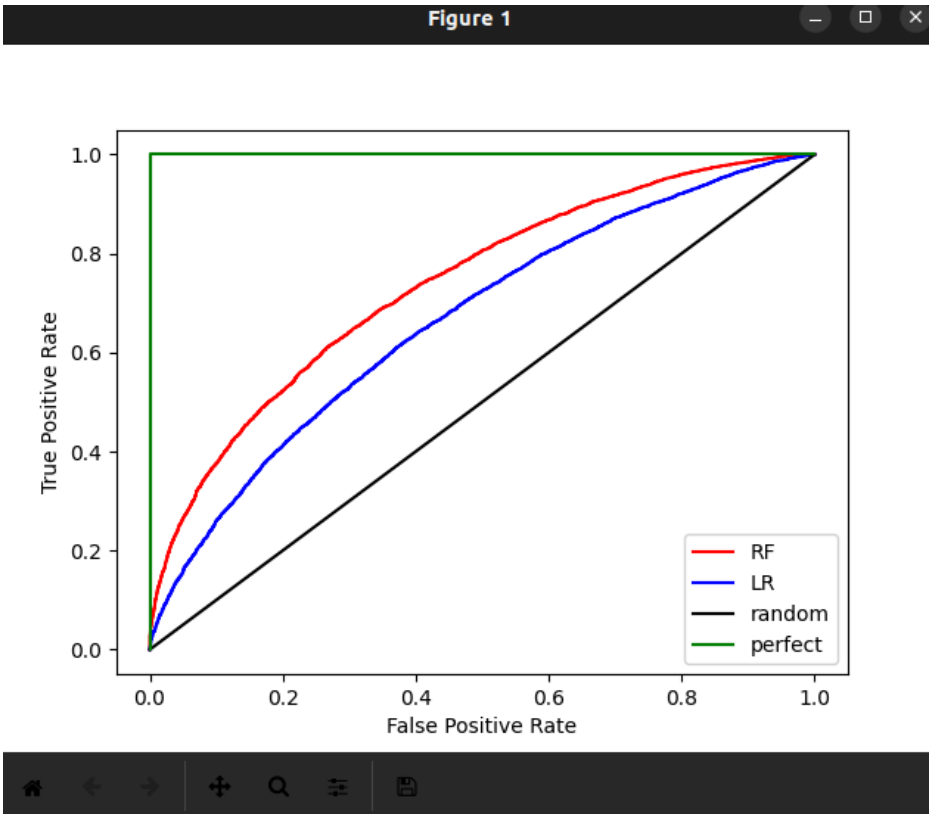
Поріг 0.5:

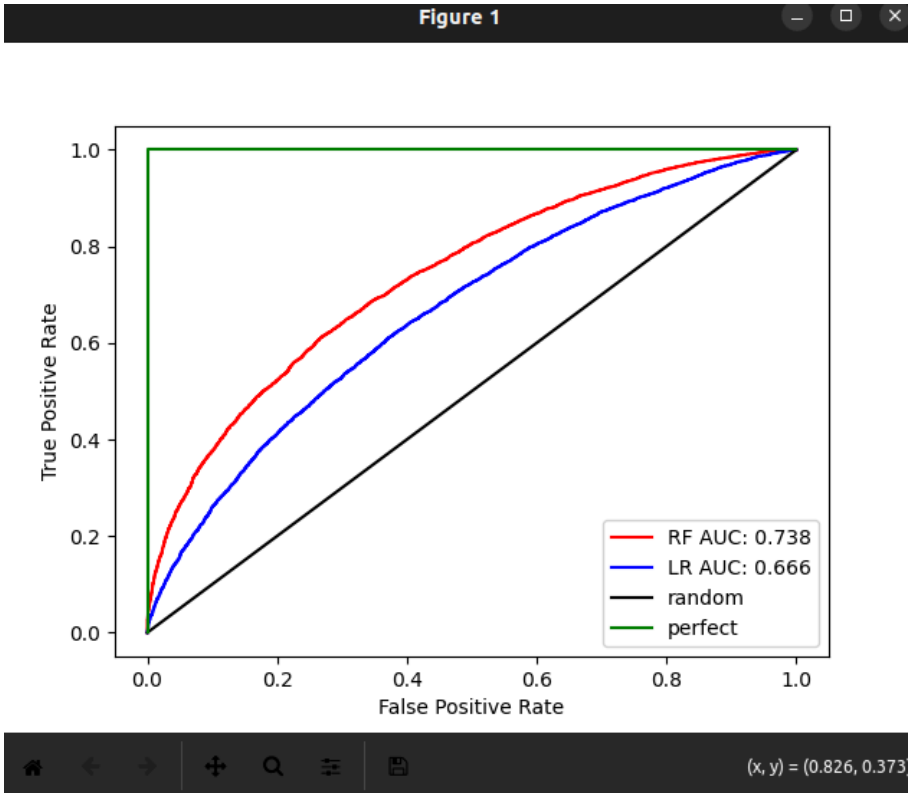
- Може забезпечити більш збалансовані результати. Краща точність, але відгук може бути нижчим.

Поріг 0.25:

- Може збільшити відгук жертвуючи точністю.

Загалом вибір відгуку має залежати від задачі та цілей, для деяких задач нижчий поріг може бути кращим, для інших кращим може бути вищий поріг.





Обираючи між RF та LR, важливо оцінити дані та цілі вашого аналізу. Серед цих моделей немає гіршої або кращої, кожна з них підходить для своїх цілей.

Якщо ваша задача вимагає простоти і дані мають лінійний характер, то LR може бути кращим вибором.

Якщо ж ваші дані є складними, містять багато взаємозв'язків і ви хочете досягти кращої точності, то RF зазвичай покаже кращі результати.

Лістинг:

```
from sklearn.metrics import f1_score

from sklearn.metrics import precision_score

from sklearn.metrics import recall_score

from sklearn.metrics import accuracy_score

from sklearn.metrics import confusion_matrix

import pandas as pd

import numpy as np



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("Matrix sklearn.metrics:", confusion_matrix(

   df.actual_label.values, df.predicted_RF.values))



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):
    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 yanushevych_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("Marix Yanushevych",
yanushevych_confusion_matrix(df.actual_label.values,
df.predicted_RF.values))
print("")
print("")


assert np.array_equal(
    yanushevych_confusion_matrix(df.actual_label.values,
df.predicted_RF.values),
    confusion_matrix(df.actual_label.values, df.predicted_RF.values)
```

```
), 'yanushevych_confusion_matrix() is not correct for RF'


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


from sklearn.metrics import accuracy_score
print("sklearn.metrics accuracy_score",
accuracy_score(df.actual_label.values, df.predicted_RF.values))


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


assert yanushevych_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 RF'


assert yanushevych_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' %
(yanushevych_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
```

```
print('Accuracy LR: %.3f' %
(yanushevych_accuracy_score(df.actual_label.values,
df.predicted_LR.values)))

print("")

print("")



from sklearn.metrics import recall_score

print("sklearn.metrics recall_score", recall_score(df.actual_label.values,
df.predicted_RF.values))



def yanushevych_recall_score(y_true, y_pred):

   TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)

   recall = TP / (TP + FN) if (TP + FN) > 0 else 0

   return recall



assert yanushevych_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 yanushevych_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' % (yanushevych_recall_score(df.actual_label.values,
df.predicted_RF.values)))

print('Recall LR: %.3f' % (yanushevych_recall_score(df.actual_label.values,
df.predicted_LR.values)))

print("")

print("")



from sklearn.metrics import precision_score
```

```
print("sklearn.metrics precision_score",
precision_score(df.actual_label.values, df.predicted_RF.values))



from sklearn.metrics import precision_score



def yanushevych_precision_score(y_true, y_pred):

   TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)

   precision = TP / (TP + FP) if (TP + FP) > 0 else 0

   return precision



assert yanushevych_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 yanushevych_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' %
(yanushevych_precision_score(df.actual_label.values,
df.predicted_RF.values)))

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

print("")

print("")



from sklearn.metrics import f1_score

print("sklearn.metrics f1_score", f1_score(df.actual_label.values,
df.predicted_RF.values))



from sklearn.metrics import f1_score
```

```
def yanushevych_f1_score(y_true, y_pred):

   recall = yanushevych_recall_score(y_true, y_pred)

   precision = yanushevych_precision_score(y_true, y_pred)

   f1 = 2 * (precision * recall) / (precision + recall) if (precision +
recall) > 0 else 0

   return f1



assert yanushevych_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 yanushevych_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' % (yanushevych_f1_score(df.actual_label.values,
df.predicted_RF.values)))

print('F1 LR: %.3f' % (yanushevych_f1_score(df.actual_label.values,
df.predicted_LR.values)))

print("")

print("")



print('scores with threshold = 0.5')

print('Accuracy RF: %.3f' %
(yanushevych_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))

print('Recall RF: %.3f' % (yanushevych_recall_score(df.actual_label.values,
df.predicted_RF.values)))

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

print('F1 RF: %.3f' % (yanushevych_f1_score(df.actual_label.values,
df.predicted_RF.values)))
```

```
print('')



print('scores with threshold = 0.25')

print('Accuracy RF: %.3f' %
(yanushevych_accuracy_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))

print('Recall RF: %.3f' % (yanushevych_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))

print('Precision RF: %.3f' %
(yanushevych_precision_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))

print('F1 RF: %.3f' % (yanushevych_f1_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))



from sklearn.metrics import roc_curve

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)



import matplotlib.pyplot as plt

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()
```

```
from sklearn.metrics import roc_auc_score

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)



import matplotlib.pyplot as plt

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()
```

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:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.metrics import classification_report, confusion_matrix


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)

svm_y_pred = svm_model.predict(X_test)


nb_model = GaussianNB()

nb_model.fit(X_train, y_train)

nb_y_pred = nb_model.predict(X_test)


print("SVM - Матриця плутанини:")

print(confusion_matrix(y_test, svm_y_pred))

print("\nSVM - Звіт про класифікацію:")

print(classification_report(y_test, svm_y_pred))


print("Naive Bayes - Матриця плутанини:")

print(confusion_matrix(y_test, nb_y_pred))

print("\nNaive Bayes - Звіт про класифікацію:")

print(classification_report(y_test, nb_y_pred))
```

```
SVM - Матриця плутанини:
[[22  0  0  0]
 [ 0 25  0  0]
 [ 0  0 21  1]
 [ 0  0  0 11]]

SVM - Звіт про класифікацію:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        22
           1       1.00      1.00      1.00        25
           2       1.00      0.95      0.98        22
           3       0.92      1.00      0.96        11

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.98        80
weighted avg       0.99      0.99      0.99        80

Naive Bayes - Матриця плутанини:
[[22  0  0  0]
 [ 0 25  0  0]
 [ 0  0 21  1]
 [ 0  0  0 11]]

Naive Bayes - Звіт про класифікацію:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        22
           1       1.00      1.00      1.00        25
           2       1.00      0.95      0.98        22
           3       0.92      1.00      0.96        11

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.98        80
weighted avg       0.99      0.99      0.99        80
```

У висновку можна сказати що конкретно з цими даними дві моделі показали однакові результати. Але знову ж, загалом, одна з цих моделей може краще підходити інша гірше і навпаки, все залежить від ваших даних та задачі.

SVM має більше налаштувань і може краще справлятися з неявними залежностями в даних, тоді як наївний баєсовський класифікатор виходить з припущення про незалежність ознак.

SVM зазвичай є більш складною моделлю в плані обчислень і налаштувань, ніж наївний баєсовський класифікатор, який є простим і швидким у навчанні.

В нашому випадку: Якщо результати однакові, вибір моделі може базуватися на простоті реалізації та швидкості навчання. У такому випадку наївний баєсовський класифікатор може бути кращим вибором.

Git: https://github.com/Alhim616/AI_Labs_Yanushevych