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 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:

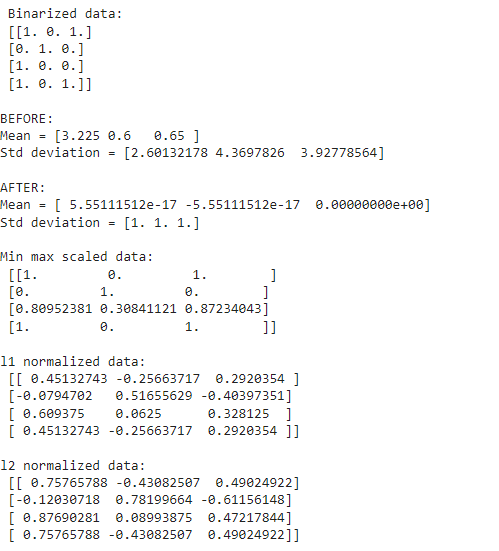


Рис. 1.

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:

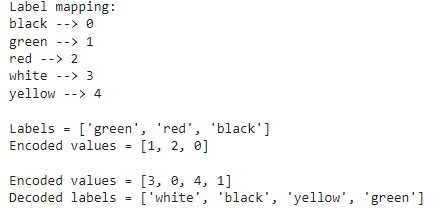
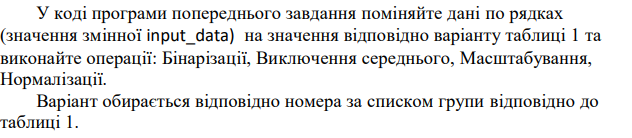


Рис. 2

Завдання 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 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:

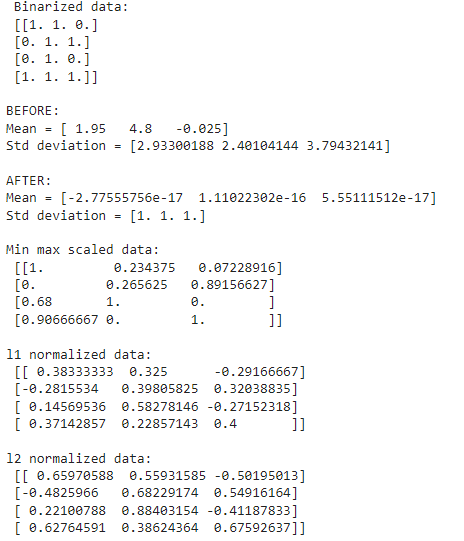


Рис. 3

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:

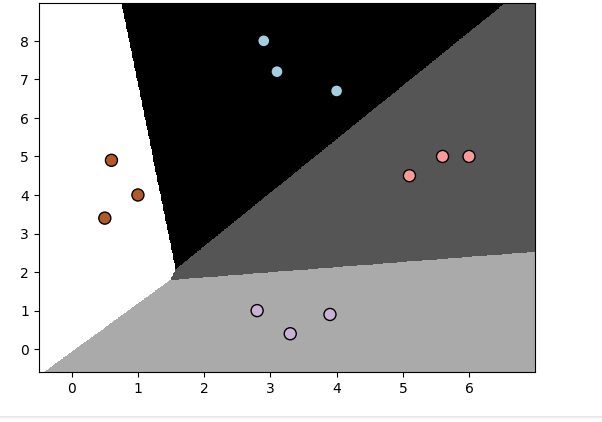
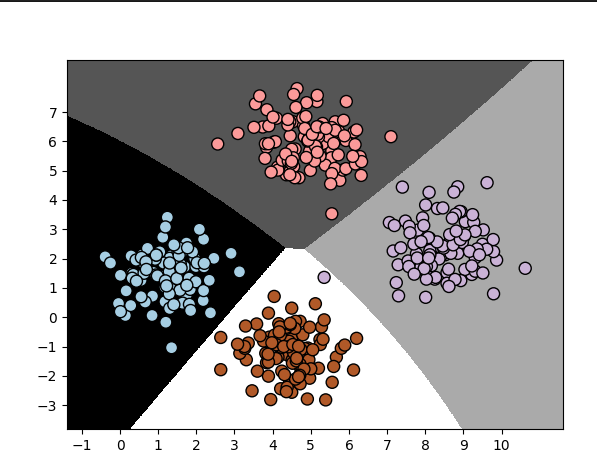


Рис. 4.

**Завдання 2.4. Класифікація наївним байєсовським класифікатором**



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:

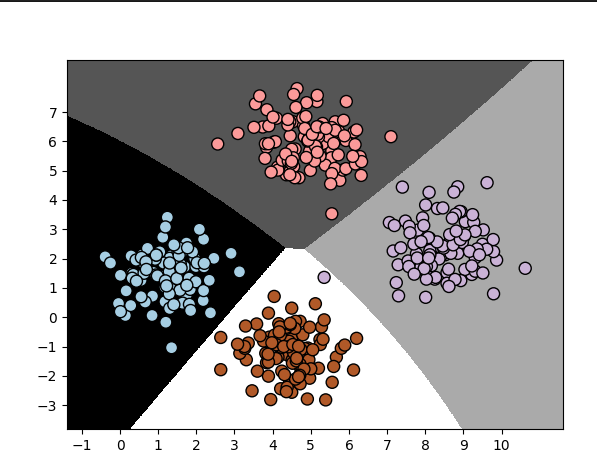
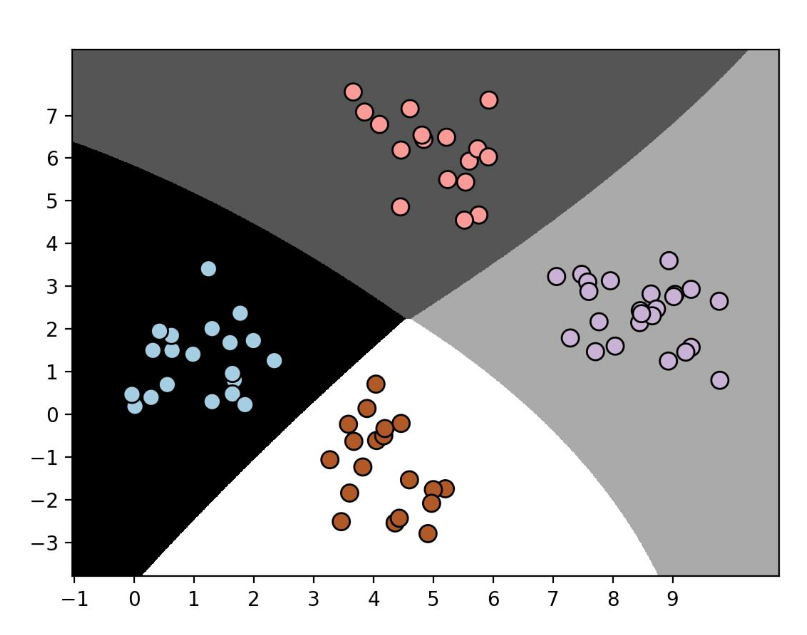


Рис. 6

Результат виконання:



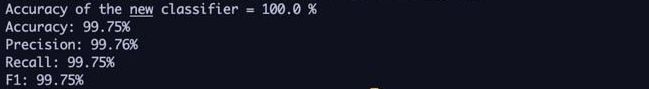
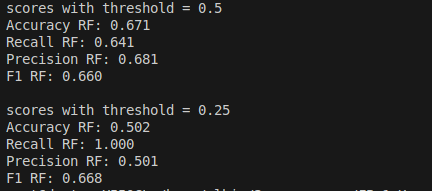


Рис. 7

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.





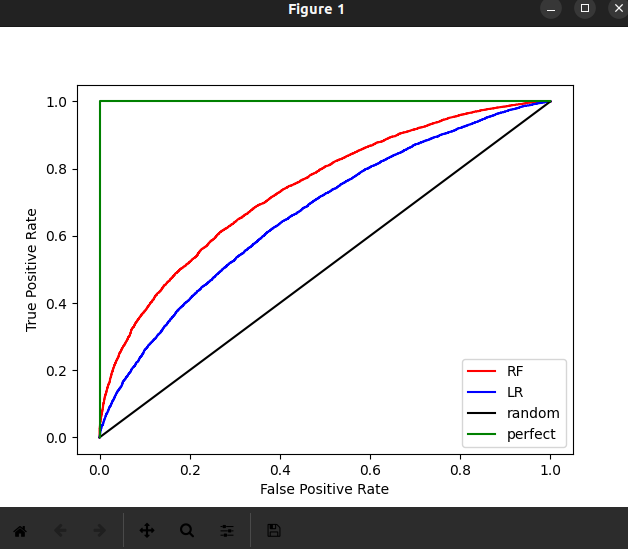
**Поріг 0.5:**

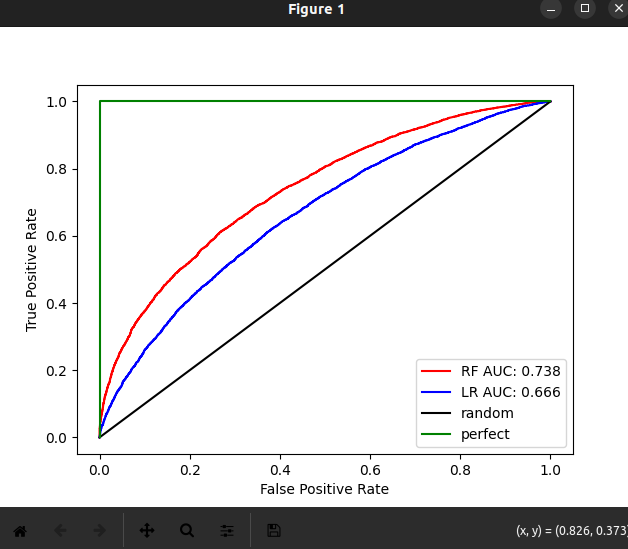
* Може забезпечити більш збалансовані результати. Краща точність, але відгук може бути нижчим.

**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')

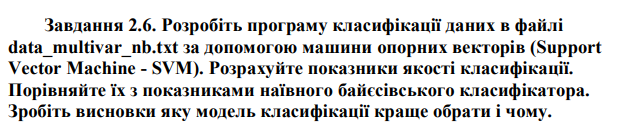
plt.plot([0,0,1,1],[0,1,1,1],'g-',label='perfect')

plt.legend()

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()



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:")

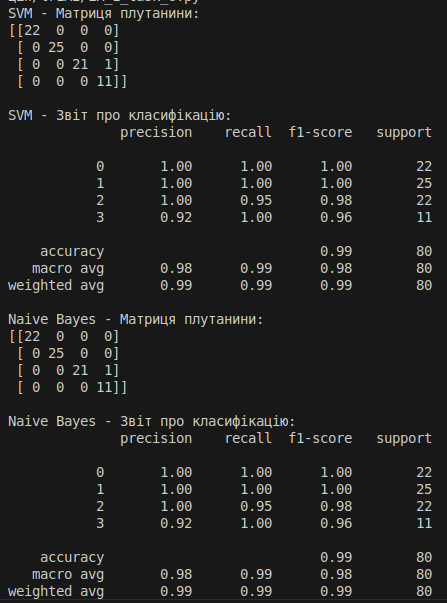
print(classification\_report(y\_test, svm\_y\_pred))

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))



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