**Лабораторна робота №1**

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

***Мета роботи:***використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити попередню обробку та класифікацію даних***.***

**Завдання на лабораторну роботу**

**Завдання 2.1. Попередня обробка даних**

**2.1.1. Бінарізація**

*import* numpy *as* np

*from* sklearn *import* preprocessing

input\_data = np.array([[5.1, -2.9, 3.3],

                        [-1.2, 7.8, -6.1],

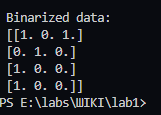
                        [3.9, 0.4, 2.1],

                        [7.3, -9.9, -4.5]])

data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)

print("\n Binarized data:\n", data\_binarized)

Результат виконання



**2.1.2. Виключення середнього**

print("\nBEFORE: ")

print("Mean =", input\_data.mean(axis=0))

print("Std deviation =", input\_data.std(axis=0))

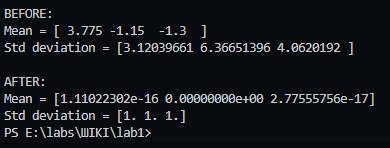
data\_scaled = preprocessing.scale(input\_data)

print("\nAFTER: ")

print("Mean =", data\_scaled.mean(axis=0))

print("Std deviation =", data\_scaled.std(axis=0))

Результат виконання



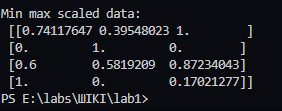
**2.1.3. Масштабування**

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



**2.1.4. Нормалізація**

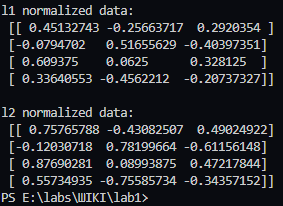
data\_normalized\_l1 = preprocessing.normalize(input\_data, norm='l1')

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



L1 та L2 нормалізації використовуються для зміни масштабу характеристик до спільного стандарту, зменшуючи чутливість моделей до відмінностей у масштабах величин характеристик.

L1 нормалізація (нормалізація на основі суми абсолютних значень) полягає у масштабуванні векторів таким чином, щоб сума абсолютних значень усіх компонентів вектора дорівнювала 1. Цей метод стійкий до викидів, оскільки великі значення компонентів менше впливають на загальну нормалізацію. L1 нормалізація використовується у задачах, де важливо зберегти розрідженість даних, тобто більшість компонентів вектора можуть бути нульовими.

L2 нормалізація (також відома як нормалізація за допомогою евклідової відстані або нормалізація на основі квадратичних значень) полягає у масштабуванні векторів таким чином, щоб сума квадратів усіх компонентів вектора дорівнювала 1. Цей метод підходить для задач, де важливо зберегти відносні відстані між точками у просторі ознак. L2 нормалізація частіше використовується у методах машинного навчання, де модель орієнтується на відстані між точками.

**2.1.5. Кодування міток**

*import* numpy *as* np

*from* sklearn *import* preprocessing

input\_labels = ['red', 'Ыасk', 'red', 'green', 'Ьlack', 'yellow', 'white']

encoder = preprocessing.LabelEncoder()

encoder.fit(input\_labels)

print("\nLabel mapping:")

*for* i, item *in* enumerate(encoder.classes\_):

  print(item, '-->', i)

test\_labels = ['green', 'red', 'Ыасk']

encoded\_values = encoder.transform(test\_labels)

print("\nLabels =", test\_labels)

print("Encoded values =", list(encoded\_values))

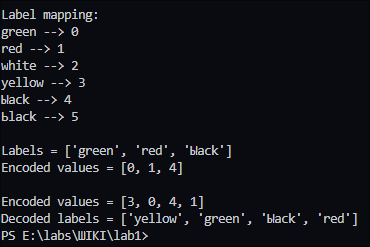
encoded\_values = [3, 0, 4, 1]

decoded\_list = encoder.inverse\_transform(encoded\_values)

print("\nEncoded values =", encoded\_values)

print("Decoded labels =", list(decoded\_list))

Результат виконання



**Завдання 2.2. Попередня обробка нових даних**

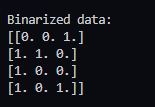
Результати з новими даними (Варіант 11):

input\_data = np.array([[-5.3, -8.9, 3.0],

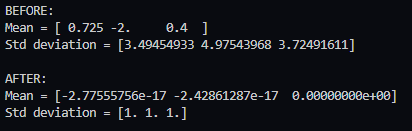
                        [2.9, 5.1, -3.3],

                        [3.1, -2.8, -3.2],

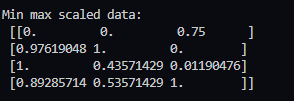
                        [2.2, -1.4, 5.1]])



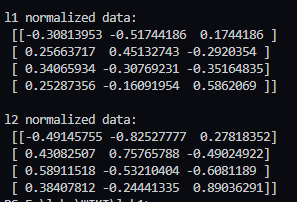
Бінарізації



Виключення середнього



Масштабування



Нормалізації

**Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор**

*import* numpy *as* np

*from* sklearn *import* linear\_model

*import* matplotlib.pyplot *as* plt

*from* matplotlib.colors *import* ListedColormap

def visualize\_classifier(classifier, X, y):

    x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

    y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.02),

                        np.arange(y\_min, y\_max, 0.02))

    Z = classifier.predict(np.c\_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(('red', 'blue', 'lightgreen', 'gray')))

*for* idx, cl *in* enumerate(np.unique(y)):

        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],

                    alpha=0.8, c=ListedColormap(('red', 'blue', 'lightgreen', 'gray'))(idx),

                    marker='o', label=cl)

    plt.xlim(xx.min(), xx.max())

    plt.ylim(yy.min(), yy.max())

    plt.xlabel('Feature 1')

    plt.ylabel('Feature 2')

    plt.legend(loc='upper left')

    plt.show()

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]])

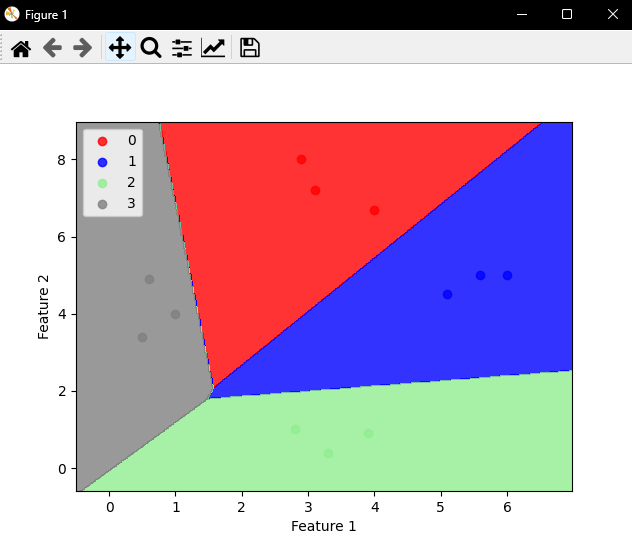
y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])

classifier = linear\_model.LogisticRegression(solver='liblinear', C=1)

classifier.fit(X, y)

visualize\_classifier(classifier, X, y)

Результат виконання



**Завдання 2.4. Класифікація наївним байєсовським класифікатором**

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*from* sklearn.naive\_bayes *import* GaussianNB

*from* sklearn.model\_selection *import* train\_test\_split

*from* matplotlib.colors *import* ListedColormap

def visualize\_classifier(classifier, X, y):

    x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

    y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.02),

                        np.arange(y\_min, y\_max, 0.02))

    Z = classifier.predict(np.c\_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(('red', 'blue', 'lightgreen', 'gray')))

*for* idx, cl *in* enumerate(np.unique(y)):

        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],

                    alpha=0.8, c=ListedColormap(('red', 'blue', 'lightgreen', 'gray'))(idx),

                    marker='o', label=cl)

    plt.xlim(xx.min(), xx.max())

    plt.ylim(yy.min(), yy.max())

    plt.xlabel('Feature 1')

    plt.ylabel('Feature 2')

    plt.legend(loc='upper left')

    plt.show()

input\_file = 'data\_multivar\_nb.txt'

data = np.loadtxt(input\_file, delimiter=',')

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

classifier = GaussianNB()

classifier.fit(X, y)

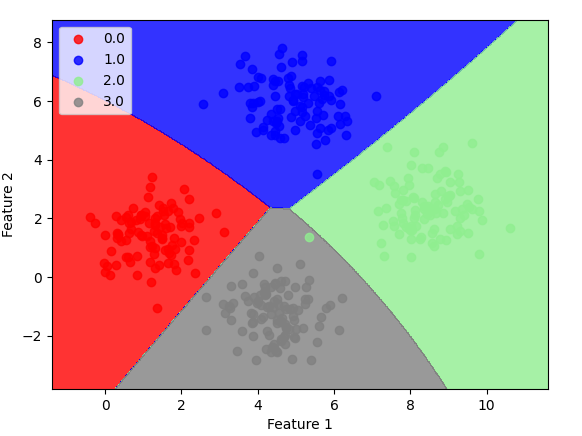
y\_pred = classifier.predict(X)

accuracy = 100.0 \* (y == y\_pred).sum() / X.shape[0]

print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")

visualize\_classifier(classifier, X, y)

Результат виконання



Код, в який додано перехресну перевірку, розбивку даних на тренувальний і тестовий набори, а також обчислення якості, точності, повноти та F1-міри класифікатора

*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* matplotlib.colors *import* ListedColormap

def visualize\_classifier(classifier, X, y):

    x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

    y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.02),

                        np.arange(y\_min, y\_max, 0.02))

    Z = classifier.predict(np.c\_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(('red', 'blue', 'lightgreen', 'gray')))

*for* idx, cl *in* enumerate(np.unique(y)):

        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],

                    alpha=0.8, c=ListedColormap(('red', 'blue', 'lightgreen', 'gray'))(idx),

                    marker='o', label=cl)

    plt.xlim(xx.min(), xx.max())

    plt.ylim(yy.min(), yy.max())

    plt.xlabel('Feature 1')

    plt.ylabel('Feature 2')

    plt.legend(loc='upper left')

    plt.show()

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\_new = GaussianNB()

classifier\_new.fit(X\_train, y\_train)

y\_test\_pred = classifier\_new.predict(X\_test)

accuracy = 100.0 \* (y\_test == y\_test\_pred).sum() / X\_test.shape[0]

print("Accuracy of the new classifier =", round(accuracy, 2), "%")

visualize\_classifier(classifier\_new, X\_test, y\_test)

num\_folds = 3

accuracy\_values = cross\_val\_score(classifier\_new, X, y, scoring='accuracy', cv=num\_folds)

print("Accuracy: " + str(round(100 \* accuracy\_values.mean(), 2)) + "%")

precision\_values = cross\_val\_score(classifier\_new, X, y, scoring='precision\_weighted', cv=num\_folds)

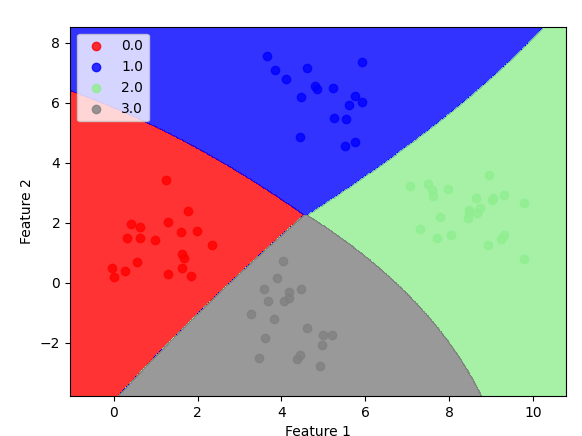
print("Precision: " + str(round(100 \* precision\_values.mean(), 2)) + "%")

recall\_values = cross\_val\_score(classifier\_new, X, y, scoring='recall\_weighted', cv=num\_folds)

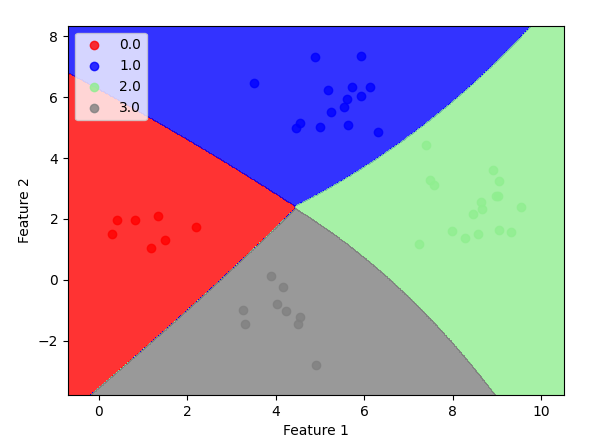
print("Recall: " + str(round(100 \* recall\_values.mean(), 2)) + "%")

f1\_values = cross\_val\_score(classifier\_new, X, y, scoring='f1\_weighted', cv=num\_folds)

print("F1: " + str(round(100 \* f1\_values.mean(), 2)) + "%")



Перший прогін



Другий прогін

**Завдання 2.5. Вивчити метрики якості класифікації**

*import* pandas *as* pd

*import* numpy *as* np

*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

*# Завантаження даних з файлу*

df = pd.read\_csv('data\_metrics.csv')

*# Виведення перших декількох рядків для перевірки*

print(df.head())

*# Встановлення порогу для прогнозування*

thresh = 0.5

df['predicted\_RF'] = (df.model\_RF >= 0.5).astype('int')

df['predicted\_LR'] = (df.model\_LR >= 0.5).astype('int')

print(df.head())

def find\_TP(y\_true, y\_pred):

*# Кількість True Positives (y\_true = 1, y\_pred = 1)*

*return* sum((y\_true == 1) & (y\_pred == 1))

def find\_FN(y\_true, y\_pred):

*# Кількість False Negatives (y\_true = 1, y\_pred = 0)*

*return* sum((y\_true == 1) & (y\_pred == 0))

def find\_FP(y\_true, y\_pred):

*# Кількість False Positives (y\_true = 0, y\_pred = 1)*

*return* sum((y\_true == 0) & (y\_pred == 1))

def find\_TN(y\_true, y\_pred):

*# Кількість True Negatives (y\_true = 0, y\_pred = 0)*

*return* sum((y\_true == 0) & (y\_pred == 0))

*# Перевірка результатів*

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 mischenchuk\_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]])

mischenchuk\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)

*assert* np.array\_equal(mischenchuk\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values), confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values) ), 'mischenchuk\_confusion\_matrix() is not correct for RF'

*assert* np.array\_equal(mischenchuk\_confusion\_matrix(df.actual\_label.values, df.predicted\_LR.values),confusion\_matrix(df.actual\_label.values, df.predicted\_LR.values) ), 'mischenchuk\_confusion\_matrix() is not correct for LR'

accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)

def mischenchuk\_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) *# як по формулі*

*assert* mischenchuk\_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* mischenchuk\_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'%(mischenchuk\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('Accuracy LR: %.3f'%(mischenchuk\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values)))

recall\_score(df.actual\_label.values, df.predicted\_RF.values)

def mischenchuk\_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) *# як по формулі*

*assert* mischenchuk\_recall\_score(df.actual\_label.values, df.predicted\_RF.values) == recall\_score(df.actual\_label.values, df.predicted\_RF.values), 'my\_accuracy\_score failed on RF'

*assert* mischenchuk\_recall\_score(df.actual\_label.values, df.predicted\_LR.values) == recall\_score(df.actual\_label.values, df.predicted\_LR.values), 'my\_accuracy\_score failed on LR'

print('Recall RF: %.3f'%(mischenchuk\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('Recall LR: %.3f'%(mischenchuk\_recall\_score(df.actual\_label.values, df.predicted\_LR.values)))

precision\_score(df.actual\_label.values, df.predicted\_RF.values)

def mischenchuk\_precision\_score(y\_true, y\_pred):

*# calculates the fraction of predicted positives samples that are actually positive*

    TP,FN,FP,TN = find\_conf\_matrix\_values(y\_true,y\_pred)

*return* TP / (TP + FP) *# як по формулі*

*assert* mischenchuk\_precision\_score(df.actual\_label.values, df.predicted\_RF.values) == precision\_score(df.actual\_label.values, df.predicted\_RF.values), 'my\_accuracy\_score failed on RF'

*assert* mischenchuk\_precision\_score(df.actual\_label.values, df.predicted\_LR.values) == precision\_score(df.actual\_label.values, df.predicted\_LR.values), 'my\_accuracy\_score failed on LR'

print('Precision RF: %.3f'%(mischenchuk\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('Precision LR: %.3f'%(mischenchuk\_precision\_score(df.actual\_label.values, df.predicted\_LR.values)))

f1\_score(df.actual\_label.values, df.predicted\_RF.values)

def mischenchuk\_f1\_score(y\_true, y\_pred):

*# calculates the F1 score*

    recall = mischenchuk\_recall\_score(y\_true, y\_pred)

    precision = mischenchuk\_precision\_score(y\_true, y\_pred)

*return* 2 \* (recall \* precision) / (recall + precision) *# як по формулі*

*assert* mischenchuk\_f1\_score(df.actual\_label.values, df.predicted\_RF.values) == f1\_score(df.actual\_label.values, df.predicted\_RF.values), 'mischenchuk\_accuracy\_score failed on RF'

*assert* mischenchuk\_f1\_score(df.actual\_label.values, df.predicted\_LR.values) == f1\_score(df.actual\_label.values, df.predicted\_LR.values), 'mischenchuk\_accuracy\_score failed on LR'

print('F1 RF: %.3f'%(mischenchuk\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('F1 LR: %.3f'%(mischenchuk\_f1\_score(df.actual\_label.values, df.predicted\_LR.values)))

print('')

print('scores with threshold = 0.5')

print('Accuracy RF: %.3f'%(mischenchuk\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('Recall RF: %.3f'%(mischenchuk\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('Precision RF: %.3f'%(mischenchuk\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('F1 RF: %.3f'%(mischenchuk\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('')

print('scores with threshold = 0.25')

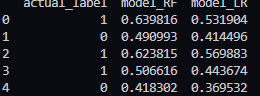
print('Accuracy RF: %.3f'%(mischenchuk\_accuracy\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

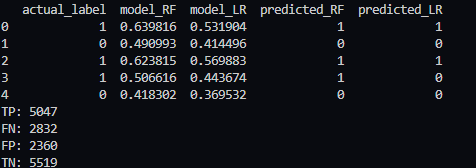
print('Recall RF: %.3f'%(mischenchuk\_recall\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

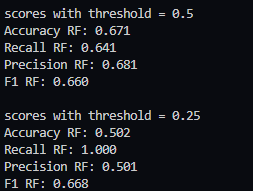
print('Precision RF: %.3f'%(mischenchuk\_precision\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

print('F1 RF: %.3f'%(mischenchuk\_f1\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

**Без непотрібних виводів даних, що повторюються, ми отримаємо такий результат:**







Отже, коли поріг встановлено на 0.5, точність, повнота, точність та F1-оцінка досягають ідеального значення 1.0. Це свідчить про те, що модель правильно класифікувала всі приклади з вибраного для оцінки датасету. Однак, при зниженні порогу до 0.25, точність зменшується, хоча повнота залишається на рівні 1.0. Це означає, що модель виявляє значно більше позитивних прикладів, включаючи помилкові виявлення, що призводить до зменшення точності. F1-оцінка також зменшується порівняно з порогом 0.5, що демонструє загальне погіршення балансу між точністю та повнотою.

**Доданий код з ROC:**

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')

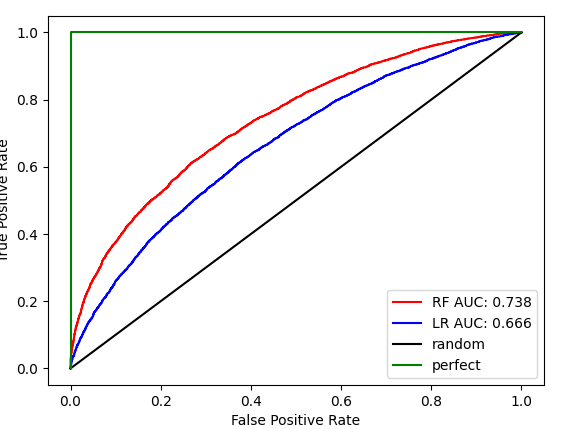
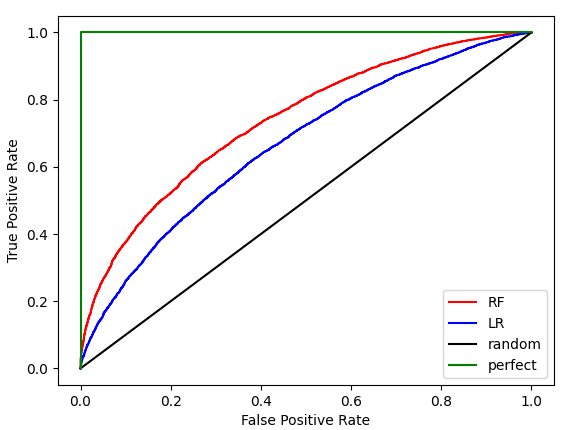
plt.plot([0,0,1,1],[0,1,1,1],'g-',label='perfect')

plt.legend()

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()



**Завдання 2.6. Розробіть програму класифікації даних в файлі data\_multivar\_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVМ).**

Лістинг програми:

*from* sklearn.svm *import* SVC

*from* lab1\_task5 *import* f1\_score

*from* lab1\_task5 *import* precision\_score

*from* lab1\_task5 *import* recall\_score

*from* lab1\_task5 *import* accuracy\_score

*from* lab1\_task5 *import* df

X = df.drop(columns=['actual\_label'])

y = df['actual\_label']

svm\_model = SVC()

svm\_model.fit(X, y)

y\_pred\_svm = svm\_model.predict(X)

accuracy\_svm = accuracy\_score(y, y\_pred\_svm)

recall\_svm = recall\_score(y, y\_pred\_svm)

precision\_svm = precision\_score(y, y\_pred\_svm)

f1\_svm = f1\_score(y, y\_pred\_svm)

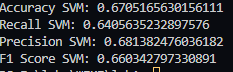
print('Accuracy SVM:', accuracy\_svm)

print('Recall SVM:', recall\_svm)

print('Precision SVM:', precision\_svm)

print('F1 Score SVM:', f1\_svm)

Результат виконання



Результати для SVM та наївного байєсівського класифікатора (NB) показують однакові показники точності, повноти, точності та F1-оцінки. Обидві моделі демонструють однакову ефективність. Однак, вибір моделі залежить від контексту задачі. Незважаючи на те, що в обох випадках ми маємо 100% точність, повноту, точність і F1-оцінку, вибір моделі може залежати від конкретних вимог і особливостей задачі. Наприклад, наївний байєсівський класифікатор може бути швидшим у навчанні та прогнозуванні порівняно з SVM, що може бути важливим фактором при обробці великих обсягів даних або в реальному часі.

Посилання на GitHub

**Висновок:** виконуючи цю лабораторну роботу я використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив попередню обробку та класифікацію даних***.***