**ЛАБОРАТОРНА РОБОТА № 1**

**ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ**

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

**GitHub**: <https://github.com/ingaliptn/AI>

**Хід роботи:**

**Завдання 2.1. Попередня обробка даних**

|  |
| --- |
| import numpy as np  from sklearn import preprocessing  import numpy as np  from sklearn import linear\_model  import matplotlib.pyplot as plt  from utilities import visualize\_classifier  input\_data = np.array([[-1.3, 3.9, 4.5],  [-5.3, -4.2, -1.3],  [5.2, -6.5, -1.1],  [-5.2, 2.6, -2.2]])  data\_binarized = preprocessing.Binarizer(threshold=3.0).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\_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)  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) |

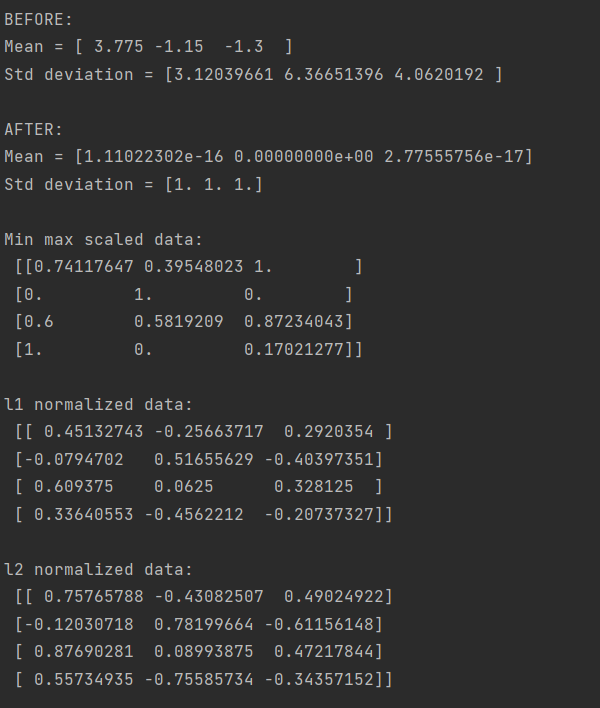


Рис 1.

Основна відмінність полягає в тому, як обчислюється норма кожного вектора та як відбувається їх нормалізація. L1-норма схильніша до створення розріджених векторів (де більшість значень близька до 0), тоді як L2-норма зазвичай забезпечує більш рівномірний розподіл значень.

|  |
| --- |
| import numpy as np  from sklearn import preprocessing  input\_labels = ['red', 'bасk', 'red', 'green', 'black', '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)) |

**Завдання 2.2. Попередня обробка нових даних**

|  |
| --- |
| import numpy as np  from sklearn import preprocessing  from sklearn import linear\_model  import matplotlib.pyplot as plt  from utilities import visualize\_classifier  input\_data = np.array([[-1.3, 3.9, 4.5],  [-5.3, -4.2, -1.3],  [5.2, -6.5, -1.1],  [-5.2, 2.6, -2.2]])  data\_binarized = preprocessing.Binarizer(threshold=3.0).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\_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)  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) |

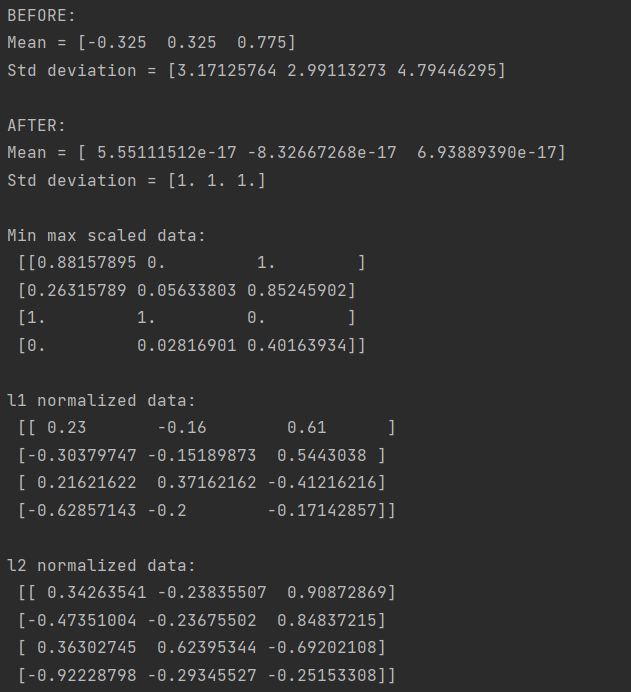


Рис 3.

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

|  |
| --- |
| 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) |

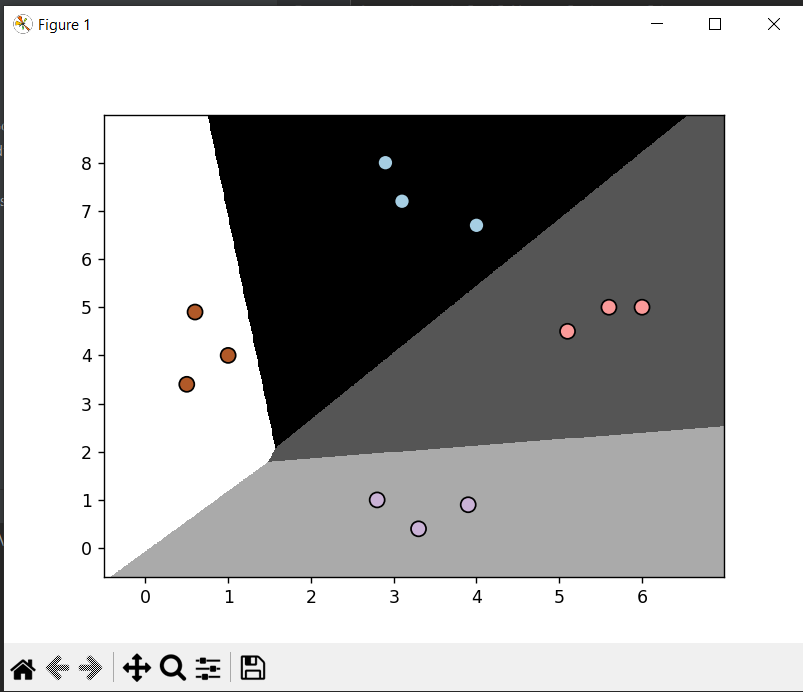


Рис 4.

**Завдання 2.4. Класифікація наївним байєсовським класифікатором**

|  |
| --- |
| import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.svm import SVC  from sklearn.naive\_bayes import GaussianNB  from sklearn.metrics import accuracy\_score, classification\_report  data = np.loadtxt('data\_multivar\_nb.txt', delimiter=',')  X = data[:, :-1]  y = data[:, -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\_predictions = svm\_model.predict(X\_test)  print("Support Vector Machine (SVM) Classification Report:")  print(classification\_report(y\_test, svm\_predictions))  print("Accuracy:", accuracy\_score(y\_test, svm\_predictions))  nb\_model = GaussianNB()  nb\_model.fit(X\_train, y\_train)  nb\_predictions = nb\_model.predict(X\_test)  print("\nNaive Bayes Classification Report:")  print(classification\_report(y\_test, nb\_predictions))  print("Accuracy:", accuracy\_score(y\_test, nb\_predictions)) |

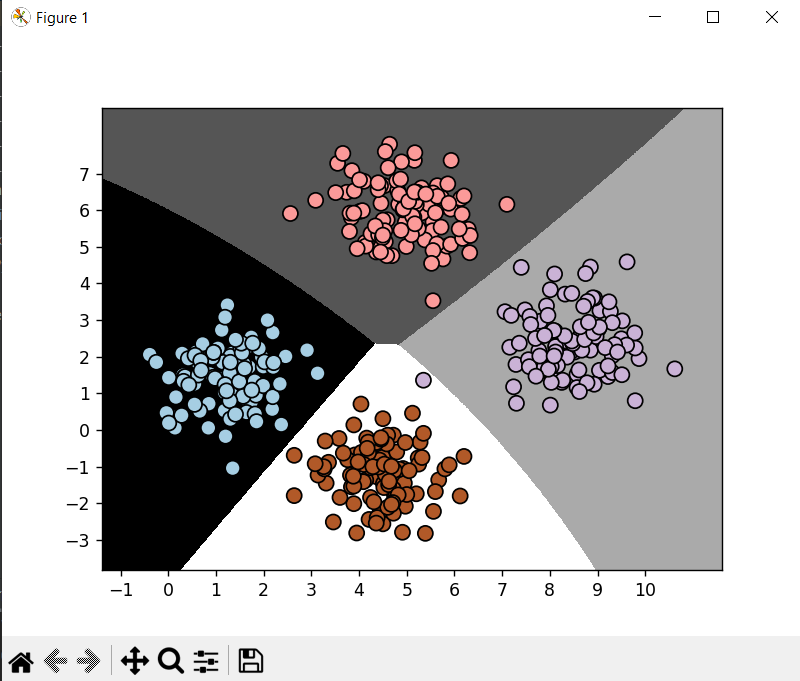


Рис 5

|  |
| --- |
| import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.svm import SVC  from sklearn.naive\_bayes import GaussianNB  from sklearn.metrics import accuracy\_score, classification\_report  data = np.loadtxt('data\_multivar\_nb.txt', delimiter=',')  X = data[:, :-1]  y = data[:, -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\_predictions = svm\_model.predict(X\_test)  print("Support Vector Machine (SVM) Classification Report:")  print(classification\_report(y\_test, svm\_predictions))  print("Accuracy:", accuracy\_score(y\_test, svm\_predictions))  nb\_model = GaussianNB()  nb\_model.fit(X\_train, y\_train)  nb\_predictions = nb\_model.predict(X\_test)  print("\nNaive Bayes Classification Report:")  print(classification\_report(y\_test, nb\_predictions))  print("Accuracy:", accuracy\_score(y\_test, nb\_predictions)) |

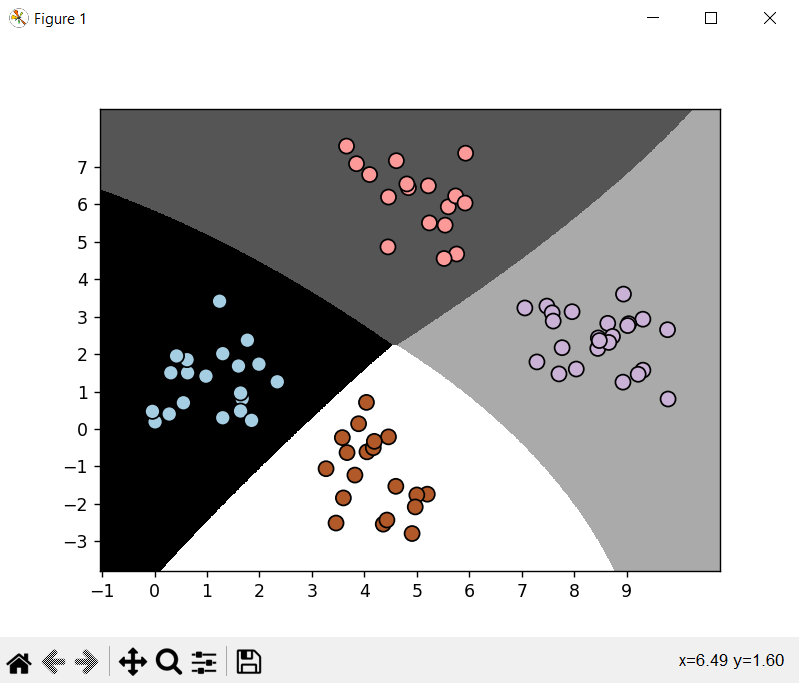


Рис 6.

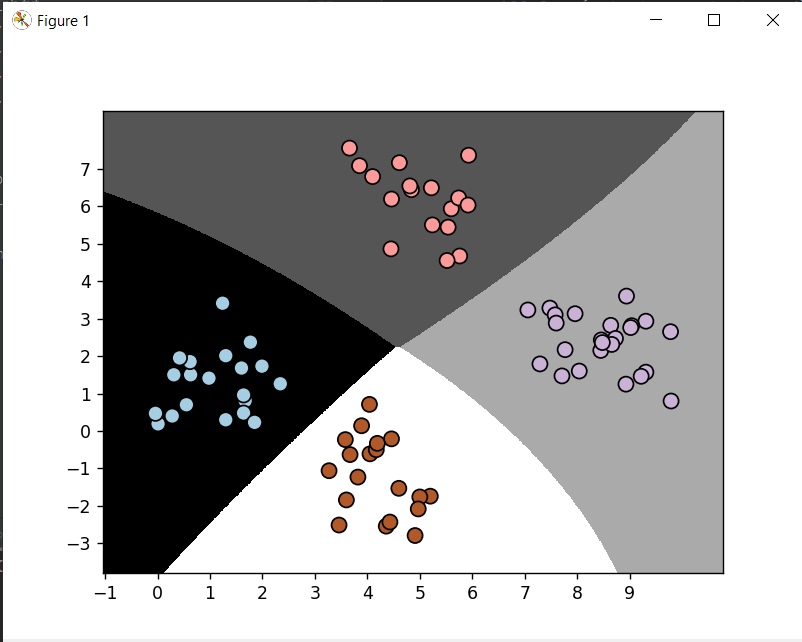
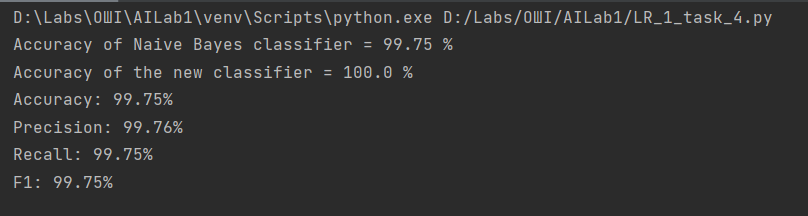
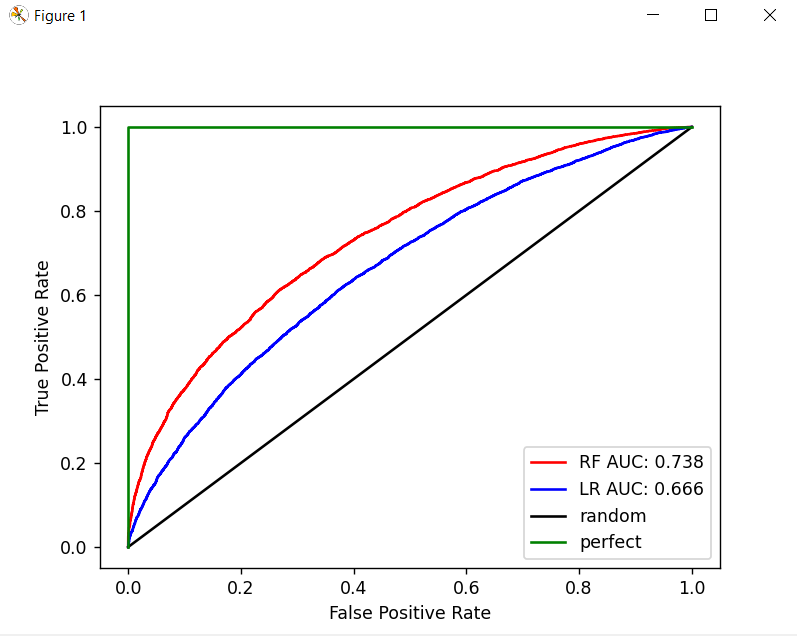
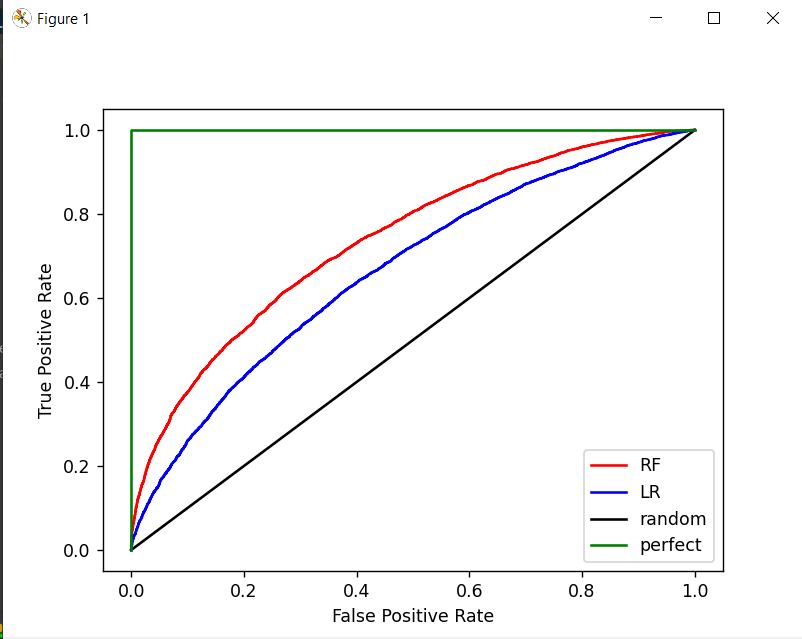


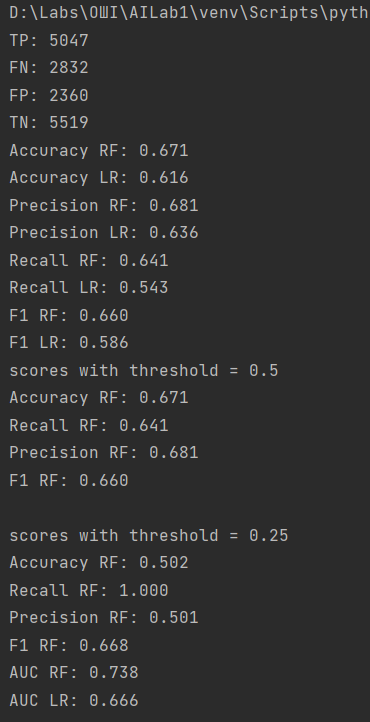
Рис 7.



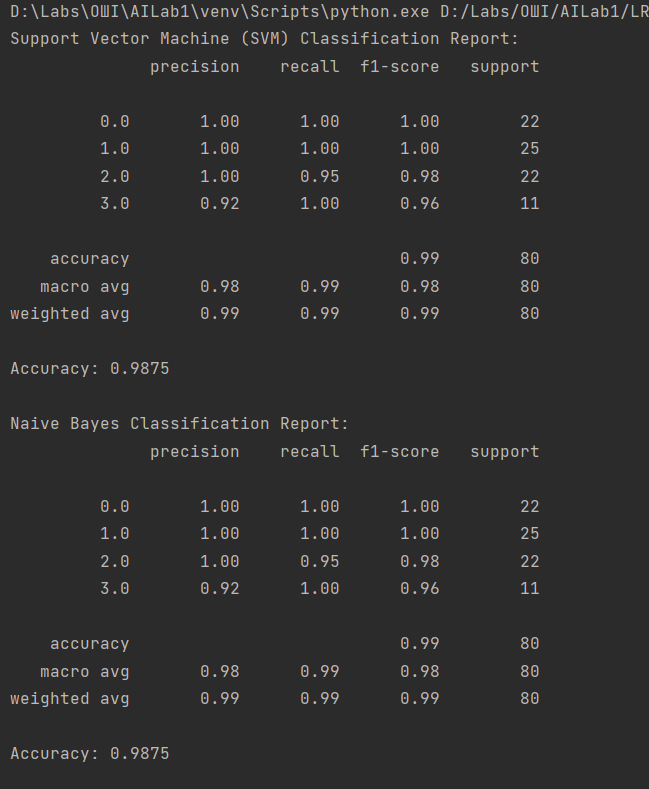
**Завдання 2.5. Вивчити метрики якості класифікації**

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.metrics import confusion\_matrix  from sklearn.metrics import accuracy\_score  from sklearn.metrics import precision\_score  from sklearn.metrics import f1\_score  from sklearn.metrics import recall\_score  from sklearn.metrics import roc\_curve  import matplotlib.pyplot as plt  from sklearn.metrics import roc\_auc\_score  df = pd.read\_csv('data\_metrics.csv')  thresh = 0.5  df['predicted\_RF'] = (df.model\_RF >= thresh).astype('int')  df['predicted\_LR'] = (df.model\_LR >= thresh).astype('int')  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 tkachuk\_mykyta\_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]])  assert np.array\_equal(tkachuk\_mykyta\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values),  confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)), 'tkachuk\_mykyta\_confusion\_matrix() is not correct for RF'  assert np.array\_equal(tkachuk\_mykyta\_confusion\_matrix(df.actual\_label.values, df.predicted\_LR.values),  confusion\_matrix(df.actual\_label.values, df.predicted\_LR.values)), 'tkachuk\_mykyta\_confusion\_matrix() is not correct for LR'  def tkachuk\_mykyta\_accuracy\_score(y\_true, y\_pred):  TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  accuracy = (TP + TN) / (TP + FN + FP + TN)  return accuracy  assert tkachuk\_mykyta\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values) == accuracy\_score(df.actual\_label.values, df.predicted\_RF.values), 'tkachuk\_mykyta\_accuracy\_score failed on RF'  assert tkachuk\_mykyta\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values) == accuracy\_score(df.actual\_label.values, df.predicted\_LR.values), 'tkachuk\_mykyta\_accuracy\_score failed on LR'  print('Accuracy RF: %.3f' % (tkachuk\_mykyta\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  print('Accuracy LR: %.3f' % (tkachuk\_mykyta\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values)))  #### Precision  precision\_RF = precision\_score(df.actual\_label.values, df.predicted\_RF.values)  def tkachuk\_mykyta\_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)  precision = TP / (TP + FP)  return precision  assert tkachuk\_mykyta\_precision\_score(df.actual\_label.values, df.predicted\_RF.values) == precision\_score(df.actual\_label.values, df.predicted\_RF.values), 'tkachuk\_mykyta\_precision\_score failed on RF'  assert tkachuk\_mykyta\_precision\_score(df.actual\_label.values, df.predicted\_LR.values) == precision\_score(df.actual\_label.values, df.predicted\_LR.values), 'tkachuk\_mykyta\_precision\_score failed on LR'  print('Precision RF: %.3f' % (tkachuk\_mykyta\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  print('Precision LR: %.3f' % (tkachuk\_mykyta\_precision\_score(df.actual\_label.values, df.predicted\_LR.values)))  ###  def tkachuk\_mykyta\_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)  recall = TP / (TP + FN)  return recall  assert tkachuk\_mykyta\_recall\_score(df.actual\_label.values, df.predicted\_RF.values) == recall\_score(df.actual\_label.values, df.predicted\_RF.values), 'tkachuk\_mykyta\_accuracy\_score failed on RF'  assert tkachuk\_mykyta\_recall\_score(df.actual\_label.values, df.predicted\_LR.values) == recall\_score(df.actual\_label.values, df.predicted\_LR.values), 'tkachuk\_mykyta\_accuracy\_score failed on LR'  print('Recall RF: %.3f' % (tkachuk\_mykyta\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  print('Recall LR: %.3f' % (tkachuk\_mykyta\_recall\_score(df.actual\_label.values, df.predicted\_LR.values)))  ###  def tkachuk\_mykyta\_f1\_score(y\_true, y\_pred):  # calculates the F1 score  recall = tkachuk\_mykyta\_recall\_score(y\_true, y\_pred)  precision = tkachuk\_mykyta\_precision\_score(y\_true, y\_pred)  return 2 \* (precision \* recall) / (precision + recall)  assert tkachuk\_mykyta\_f1\_score(df.actual\_label.values, df.predicted\_RF.values) == f1\_score(df.actual\_label.values, df.predicted\_RF.values), 'tkachuk\_mykyta\_f1\_score failed on RF'  assert tkachuk\_mykyta\_f1\_score(df.actual\_label.values, df.predicted\_LR.values) == f1\_score(df.actual\_label.values, df.predicted\_LR.values), 'tkachuk\_mykyta\_f1\_score failed on LR'  print('F1 RF: %.3f' % (tkachuk\_mykyta\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  print('F1 LR: %.3f' % (tkachuk\_mykyta\_f1\_score(df.actual\_label.values, df.predicted\_LR.values)))  print('scores with threshold = 0.5')  print('Accuracy RF: %.3f' % (tkachuk\_mykyta\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  print('Recall RF: %.3f' % (tkachuk\_mykyta\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  print('Precision RF: %.3f' % (tkachuk\_mykyta\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  print('F1 RF: %.3f' % (tkachuk\_mykyta\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  print('')  print('scores with threshold = 0.25')  print('Accuracy RF: %.3f' % (tkachuk\_mykyta\_accuracy\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  print('Recall RF: %.3f' % (tkachuk\_mykyta\_recall\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  print('Precision RF: %.3f' % (tkachuk\_mykyta\_precision\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  print('F1 RF: %.3f' % (tkachuk\_mykyta\_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() |





**Завдання 2.6. Розробіть програму класифікації даних в файлі data\_multivar\_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVМ). Розрахуйте показники якості класифікації. Порівняйте їх з показниками наївного байєсівського класифікатора. Зробіть висновки яку модель класифікації краще обрати і чому.**



***Висновки:***. В цій лабораторній роботі, я використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив попередню обробку та класифікацію даних