**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**

Завдання 1. Попередня обробка даних

Для роботи з даними необхідно використовувати спеціалізовані бібліотеки функцій. Надалі використовуються numpy та sklearn.

Лістинг коду підключень бібліотек файлу Task1.py:

*import* numpy *as* np  
*from* sklearn *import* preprocessing

Серед методів попередньої обробки даних досліджуються бінаризація, виключення середнього, масштабування, нормалізація.

Лістинг коду методів обробки файлу Task1.py:

*# Дані до обробки*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]  
])  
  
*# Бінаризація даних*data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)  
*print*(f"Binarized data:\n{data\_binarized}")  
  
*# 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))  
  
*# Масштабування*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*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*("l2 normalized data:\n", data\_normalized\_l2)

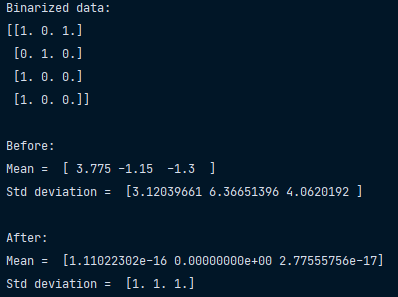


Рис.1.1 – Бінаризація та виключення середнього

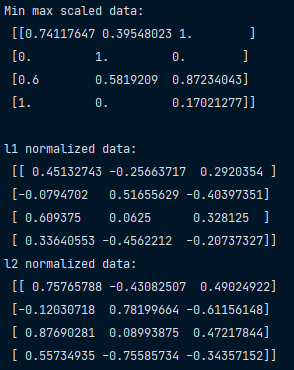


Рис.1.2 – Масштабування та нормалізація

Нормалізація L1 та L2 відрізняються точністю значень, отриманих в розрахунках суми (абсолютних значень за L1 та квадратів значень за L2). Застосування 2-го методу надає меншу точність та є менш надійним, у той час як 1-й не дозволяє вирішувати завдання, де необхідно простежувати неточність вхідних даних (викиди).

Для класифікації даних необхідно працювати з мітками, які часто для зручності є текстовими. Використовувані функції машинного навчання передбачають використання чисельних міток, через що необхідно текстові мітки перетворювати, використовуючи їх кодування.

Лістинг коду кодування міток файлу Task1.py:

*# Кодування міток*input\_labels = ['red', 'black', 'red', 'green', 'black', 'yellow', 'white']  
encoder = preprocessing.LabelEncoder()  
encoder.fit(input\_labels)  
*print*("\nLabel mapping:")  
*for* i, item *in enumerate*(encoder.classes\_):  
 *print*(f"{item} --> {i}")  
  
test\_labels = ['green', 'red', 'black']  
encoded\_values = encoder.transform(test\_labels)  
*print*("\nLabels: ", test\_labels)  
*print*("Encoded values: ", encoded\_values)  
  
encoded\_values = [3, 0, 4, 1]  
decoded\_list = encoder.inverse\_transform(encoded\_values)  
*print*("\nEncoded values: ", encoded\_values)  
*print*("Decoded labels: ", decoded\_list)

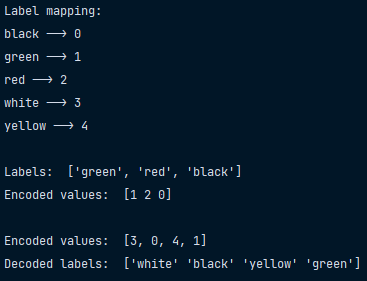


Рис.1.3 – Кодування міток

Завдання 2. Попередня обробка нових даних

Необхідно виконати операції бінаризації, виключення середнього, масштабування та нормалізації відносно нових даних власного варіанту (11й).

Лістинг коду файлу Task\_2.py:

*import* numpy *as* np  
*from* sklearn *import* preprocessing  
  
*# Дані до обробки (11й варіант)*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]  
])  
  
*# Бінаризація даних*data\_binarized = preprocessing.Binarizer(threshold=2).transform(input\_data)  
*print*(f"Binarized data:\n{data\_binarized}")  
  
*# 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))  
  
*# Масштабування*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*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*("l2 normalized data:\n", data\_normalized\_l2)

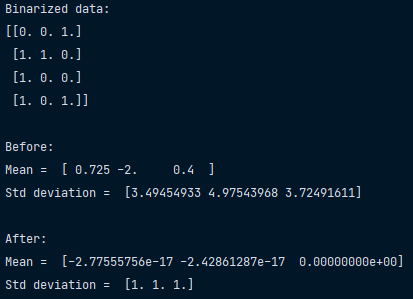


Рис.1.4 – Бінаризація та виключення середнього власних даних

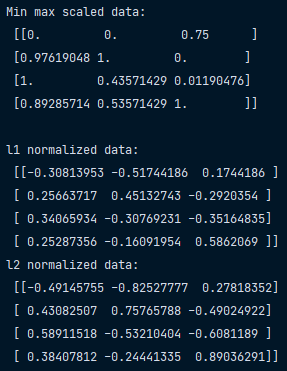


Рис.1.5 – Масштабування та нормалізація власних даних

Завдання 3. Класифікація логістичною регресією або логістичний класифікатор

Для класифікації даних, а саме спрощення цього, використовується логістична регресія. Завдяки модулю utilities.py, який було надано для виконання лабораторної роботи,

Лістинг коду файлу Task\_3.py:

*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)

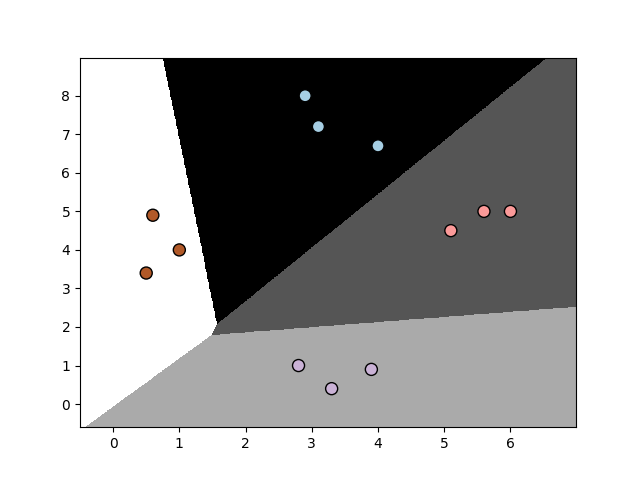


Рис.1.6 – Результат класифікації лінійною регресією

Завдання 4. Класифікація наївним байєсовським класифікатором

Наївний Байєс є набором методів класифікації, що не бере до уваги можливість залежності ознак між собою та наразі існує лише як навчальний приклад.

Лістинг коду файлу Task\_4.py:

*import* numpy *as* np  
*import* matplotlib.pyplot *as* plt  
*from* sklearn.naive\_bayes *import* GaussianNB  
*from* sklearn.model\_selection *import* train\_test\_split  
*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*(f"Accuracy of Naive Bayes classifier: {*round*(accuracy, 2)}%")  
visualize\_classifier(classifier, X, y)



Рис.1.7 – Якість класифікатора

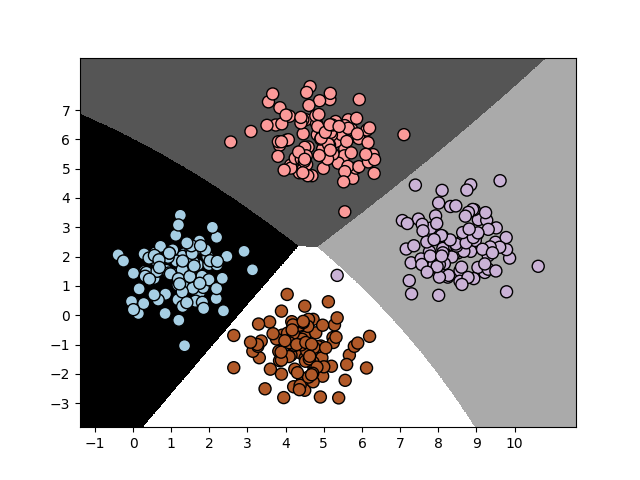


Рис.1.8 – Відображення результату класифікації

Для якісного обрахунку точності необхідно розділити дані на навчальний та тесовий набори.

Лістинг коду файлу Task\_4.py з розділенням даних:

*# Аналіз із розділенням 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)  
  
accuracy = 100 \* (y\_test == y\_test\_pred).sum() / X\_test.shape[0]  
*print*(f"Accuracy of the new Naive Bayes classifier: {*round*(accuracy, 2)}%")  
visualize\_classifier(classifier\_new, X\_test, y\_test)  
  
num\_folds = 3  
accuracy\_values = cross\_val\_score(classifier\_new, X\_test, y\_test, scoring='accuracy', cv=num\_folds)  
*print*(f"Accuracy: {*round*(100 \* accuracy\_values.mean(), 2)}%")  
  
precision\_values = cross\_val\_score(classifier\_new, X\_test, y\_test, scoring='precision\_weighted', cv=num\_folds)  
*print*(f"Precision: {*round*(100 \* precision\_values.mean(), 2)}%")  
  
recall\_values = cross\_val\_score(classifier\_new, X\_test, y\_test, scoring='recall\_weighted', cv=num\_folds)  
*print*(f"Recall: {*round*(100 \* recall\_values.mean(), 2)}%")  
  
f1\_values = cross\_val\_score(classifier\_new, X\_test, y\_test, scoring='f1\_weighted', cv=num\_folds)  
*print*(f"F1: {*round*(100 \* f1\_values.mean(), 2)}%")

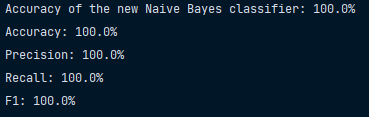


Рис.1.9 – Отримані дані про якість



Рис.1.10 – Зображення результату класифікації тестових даних

Розділення даних дозволило більш надійно отримати відповідь, а використання функції для обчислення якості, точності та повноти дозволило більш детально вказати результат.

Завдання 5. Вивчити метрики якості класифікації

Важливими метриками якості є якість, точність, чутливість та F1. Їх обчислення відбувається завдяки порівнянню результатів з реальністю, а саме зберіганням значень TP, FN, FP, TN.

Лістинг коду файлу Task\_5.py:

*import* numpy *as* np  
*import* pandas *as* pd  
*from* sklearn.metrics *import* confusion\_matrix, accuracy\_score, recall\_score, precision\_score, f1\_score, \  
 roc\_curve, roc\_auc\_score  
*import* matplotlib.pyplot *as* plt  
  
df = pd.read\_csv('data\_metrics.csv')  
*print*(df.head())  
  
thresh = 0.5  
df['predicted\_RF'] = (df.model\_RF >= thresh).astype('int')  
df['predicted\_LR'] = (df.model\_LR >= thresh).astype('int')  
*print*(df.head())  
actual = df.actual\_label.values  
model\_RF = df.model\_RF.values  
model\_LR = df.model\_LR.values  
predicted\_RF = df.predicted\_RF.values  
predicted\_LR = df.predicted\_LR.values  
  
conf\_matr = confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  
*print*("confusion\_matrix:\n", conf\_matr)  
  
  
*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):  
 *"""* ***:param*** *y\_true: List with true data of classification* ***:param*** *y\_pred: List with predicted data of classification* ***:return****: 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* Oleksiichuk\_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*("Oleksiichuk\_confusion\_matrix:\n", Oleksiichuk\_confusion\_matrix(actual, predicted\_RF))  
  
*assert* np.array\_equal(Oleksiichuk\_confusion\_matrix(actual, predicted\_RF),  
 confusion\_matrix(actual, predicted\_RF)), \  
 'my confusion\_matrix() is not correct for RF'  
  
*assert* np.array\_equal(Oleksiichuk\_confusion\_matrix(actual, predicted\_LR),  
 confusion\_matrix(actual, predicted\_LR)), \  
 'my confusion\_matrix() is not correct for lR'  
  
*# Accuracy*score = accuracy\_score(actual, predicted\_RF)  
*print*("Accuracy score on RF:", score)  
  
  
*def* Oleksiichuk\_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)  
  
  
*assert* Oleksiichuk\_accuracy\_score(actual, predicted\_RF) == accuracy\_score(actual, predicted\_RF), \  
 'my accuracy\_score failed RF'  
  
*assert* Oleksiichuk\_accuracy\_score(actual, predicted\_LR) == accuracy\_score(actual, predicted\_LR), \  
 'my accuracy\_score failed LR'  
  
*print*("My accuracy score on RF:", Oleksiichuk\_accuracy\_score(actual, predicted\_RF))  
*print*("My accuracy score on LR:", Oleksiichuk\_accuracy\_score(actual, predicted\_LR))  
  
*# Recall  
print*('Recall score on RF:', recall\_score(actual, predicted\_RF))  
  
  
*def* Oleksiichuk\_recal\_score(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 *return* TP / (TP + FN)  
  
  
*assert* Oleksiichuk\_recal\_score(actual, predicted\_RF) == recall\_score(actual, predicted\_RF),\  
 'my recal\_score fails on RF'  
  
*assert* Oleksiichuk\_recal\_score(actual, predicted\_LR) == recall\_score(actual, predicted\_LR),\  
 'my recal\_score fails on LR'  
  
*print*("My recall score on RF:", Oleksiichuk\_recal\_score(actual, predicted\_RF))  
*print*("My recall score on LR:", Oleksiichuk\_recal\_score(actual, predicted\_LR))  
  
*# Precision  
print*("Precision score on RF:", precision\_score(actual, predicted\_RF))  
  
*def* Oleksiichuk\_precision\_score(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 *return* TP / (TP + FP)  
  
  
*assert* Oleksiichuk\_precision\_score(actual, predicted\_RF) == precision\_score(actual, predicted\_RF),\  
 'my precision\_score fails on RF'  
  
*assert* Oleksiichuk\_precision\_score(actual, predicted\_LR) == precision\_score(actual, predicted\_LR),\  
 'my precision\_score fails on LR'  
  
*print*("My precision score on RF:", Oleksiichuk\_precision\_score(actual, predicted\_RF))  
*print*("My precision score on LR:", Oleksiichuk\_precision\_score(actual, predicted\_LR))  
  
*# F1 score  
print*("F1 score on RF", f1\_score(actual, predicted\_RF))  
  
  
*def* Oleksiichuk\_f1\_score(y\_true, y\_pred):  
 precision = Oleksiichuk\_precision\_score(y\_true, y\_pred)  
 recall = Oleksiichuk\_recal\_score(y\_true, y\_pred)  
 *return* (2 \* (precision \* recall)) / (precision + recall)  
  
  
*assert* Oleksiichuk\_f1\_score(actual, predicted\_RF) == f1\_score(actual, predicted\_RF),\  
 'my f1\_score fails on RF'  
  
*assert* Oleksiichuk\_f1\_score(actual, predicted\_LR) == f1\_score(actual, predicted\_LR),\  
 'my f1\_score fails on LR'  
  
*print*("My F1 score score on RF:", Oleksiichuk\_f1\_score(actual, predicted\_RF))  
*print*("My F1 score score on LR:", Oleksiichuk\_f1\_score(actual, predicted\_LR))  
*print*()  
  
  
*def* test\_thresholds(threshold: *float* = .5):  
 *print*(f"Scores with threshold = {threshold}")  
 predicted = (df.model\_RF >= threshold).astype('int')  
  
 *print*("Accuracy RF:", Oleksiichuk\_accuracy\_score(actual, predicted))  
 *print*("Precision RF:", Oleksiichuk\_precision\_score(actual, predicted))  
 *print*("Recall RF:", Oleksiichuk\_recal\_score(actual, predicted))  
 *print*("F1 RF:", Oleksiichuk\_f1\_score(actual, predicted))  
 *print*()  
  
  
test\_thresholds()  
test\_thresholds(.25)  
test\_thresholds(.6)  
test\_thresholds(.20)  
  
*# ROC  
# Curve*fpr\_RF, tpr\_RF, thresholds\_RF = roc\_curve(actual, model\_RF)  
fpr\_LR, tpr\_LR, thresholds\_LR = roc\_curve(actual, model\_LR)  
  
*# AUC*auc\_RF = roc\_auc\_score(actual, model\_RF)  
auc\_LR = roc\_auc\_score(actual, model\_LR)  
  
*print*("AUC RF:", auc\_RF)  
*print*("AUC LR:", auc\_LR)  
  
plt.plot(fpr\_RF, tpr\_RF, 'r-', label=f'AUC RF: {auc\_RF}')  
plt.plot(fpr\_LR, tpr\_LR, 'b-', label=f'AUC LR: {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()

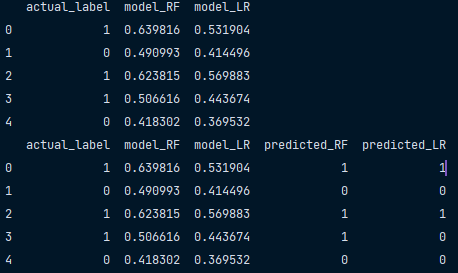


Рис.1.11 – Вхідні та прогнозовані дані, перші 5 рядків

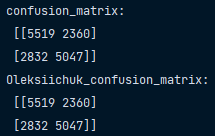


Рис.1.12 – Робота власної та наданої функцій отримання матриць помилок

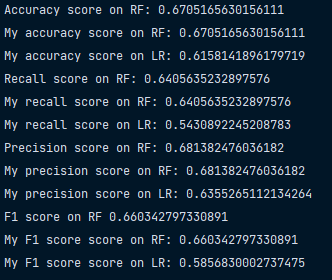


Рис.1.13 – Метрика моделей, отримана власними та наданими функціями

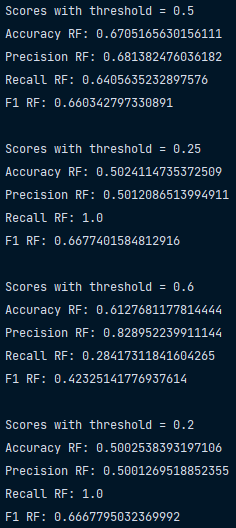


Рис.1.14 – Метрика моделі RF за різних порогів

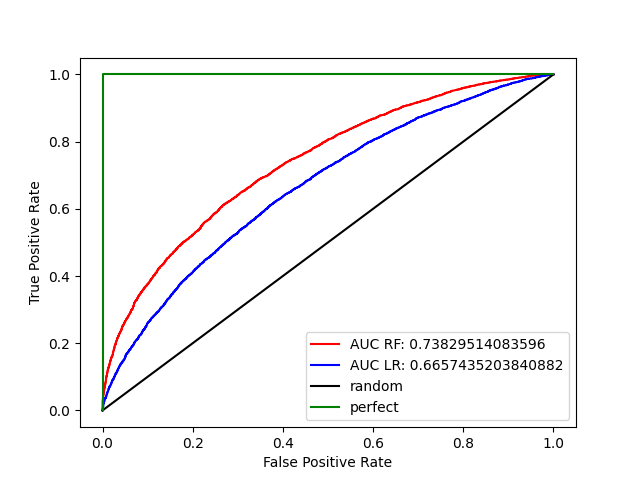


Рис.1.15 – Графік отриманих значень ROC

Завдання 6. Класифікація даних зі завдання 4 за допомоги машини опорних векторів (Support Vector Machine SVM).

Лістинг коду файлу Task\_6.py:

*import* numpy *as* np  
*from* sklearn *import* svm  
*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 = svm.SVC()  
classifier.fit(X\_train, y\_train)  
y\_test\_pred = classifier.predict(X\_test)  
  
visualize\_classifier(classifier, X\_test, y\_test)  
  
num\_folds = 3  
accuracy\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='accuracy', cv=num\_folds)  
*print*(f"Accuracy: {*round*(100 \* accuracy\_values.mean(), 2)}%")  
  
precision\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='precision\_weighted', cv=num\_folds)  
*print*(f"Precision: {*round*(100 \* precision\_values.mean(), 2)}%")  
  
recall\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='recall\_weighted', cv=num\_folds)  
*print*(f"Recall: {*round*(100 \* recall\_values.mean(), 2)}%")  
  
f1\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='f1\_weighted', cv=num\_folds)  
*print*(f"F1: {*round*(100 \* f1\_values.mean(), 2)}%")

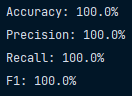


Рис.1.16 – Показники класифікації з розділенням даних на 80% навчальних

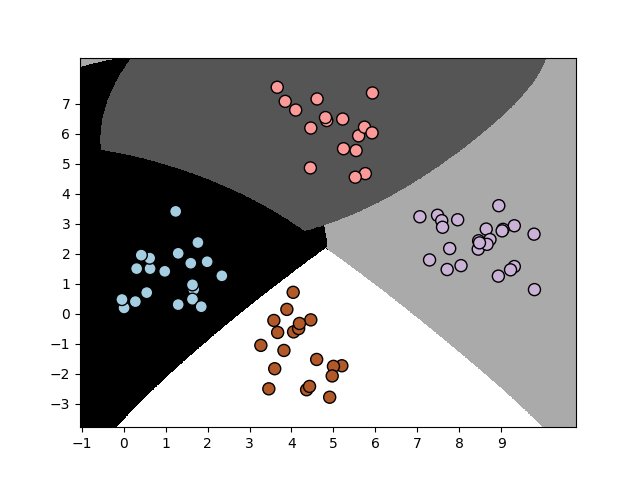


Рис.1.17 – Зображення результату класифікації тестових даних за допомоги SVM

Використання SVM надає кращі результати класифікації завдяки швидкості та простоті, проте для використання для багатокласової класифікації він не пристосований. Окрім цього, кількість даних може бути недостатньою через однакові показники.

***Висновок:*** під час виконання завдань лабораторної роботи, було отримано навички з: попередньої обробки даних шляхами бінаризації, виключення середнього, масштабування, нормалізації, кодування міток та закріплено на даних по варіантах; класифікації даних логістичною регресією; класифікації даних Наївним Байєсом; отримання та аналізу метрик якості класифікації; використання SVM та класифікації з використанням SVM даних. Під час аналізу метрик якості класифікації було розроблено власні функції з отримання необхідних даних та їх групування в матрицю помилок, порівняно отримані дані з даними від функцій.

Проект до лабораторної роботи можна переглянути за посиланням: <https://github.com/nikitoss888/AI_LR1>