Практична робота № 1

Варіант 13

Попередня обробка та контрольована

Класифікація даних

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

Хід роботи:

Завдання 1.1: Попередня обробка даних:

Як правило, при обробці ми маємо справу з великими обсягами необроблених вихідних даних. Алгоритми машинного навчання розраховані на те, що, перш ніж вони зможуть розпочати процес тренування, отримані дані будуть відформатовані певним чином. Щоб привести дані до форми, що прийнятна для алгоритмів машинного навчання, ми повинні попередньо підготувати їх і перетворити на потрібний формат.

Зробіть висновок чим відрізняються L1-нормалізація від L2-нормалізації?

Нормалізація L1 та L2 відрізняються точністю jотриманих після розрахунків суми значень. L2 має меншу точність та є менш надійним, у той час як L1 є манш універсальним та не простежувати неточність вхідних даних.

					ДУ «Житомирська політехніка».22.121.13.000 – Лр01		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Маковська О.Ю			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 10
Перевір.		Пулеко І. В.					
Керівник						ФІКТ Гр. ІПЗ-19-1[2]	
Н. контр.							
Зав. каф.							

Лістинг файлу task-1.py

```
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]
])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print(f"Binarized data:\n{data_binarized}")

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

#Виняток середнього
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)

# Кодування міток
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(f"{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: ", 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: ", decoded_list)
```

```
C:\Python311\python.exe C:/Users/Ola/Documents/stud/4/AI_Python/Lr_1/lab-1/task-1.py
Binarized data:
[[1. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]

BEFORE:
Mean =  [ 3.775 -1.15  -1.3  ]
Std deviation =  [3.12039661 6.36651396 4.0620192 ]

AFTER:
Mean =  [1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation =  [1. 1. 1.]

Min max scaled data:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]

l1 normalized data:
 [[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.0794702   0.51655629 -0.40397351]
 [ 0.609375    0.0625      0.328125  ]
 [ 0.33640553 -0.4562212  -0.20737327]]
l2 normalized data:
 [[ 0.75765788 -0.43082507  0.49024922]
 [-0.12030718  0.78199664 -0.61156148]
 [ 0.87690281  0.08993875  0.47217844]

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']

Process finished with exit code 0
```

Рис.1.1. task-1.py

Завдання 1.2: 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.

Таблиця 1

№ варіанту	Значення змінної input_data											Поріг бінаризації
13.	-2.3	-1.6	-6.1	-2.4	-1.2	4.3	3.2	3.1	6.1	-4.4	1.4	-1.2	2.1

Лістинг файлу task-2.py

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([
    [-2.3, -1.6, -6.1],
    [-2.4, -1.2, 4.3],
    [3.2, 3.1, 6.1],
    [-4.4, 1.4, -1.2]
])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print(f"Binarized data:\n{data_binarized}")

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

# Виняток середнього
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)
```

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

BEFORE:
Mean =  [-1.475  0.425  0.775]
Std deviation =  [2.82610598 1.92662269 4.79446295]

AFTER:
Mean =  [-5.55111512e-17 -5.55111512e-17 -4.16333634e-17]
Std deviation =  [1. 1. 1.]

Min max scaled data:
 [[0.27631579 0.         0.        ]
 [0.26315789 0.08510638 0.85245902]
 [1.         1.         1.        ]
 [0.         0.63829787 0.40163934]]

l1 normalized data:
 [[-0.23       -0.16       -0.61      ]
 [-0.30379747 -0.15189873  0.5443038 ]
 [ 0.25806452  0.25        0.49193548]
 [-0.62857143  0.2        -0.17142857]]
l2 normalized data:
 [[-0.34263541 -0.23835507 -0.90872869]
 [-0.47351004 -0.23675502  0.84837215]
 [ 0.42362745  0.41038909  0.80753983]
 [-0.92228798  0.29345527 -0.25153308]]
```

Рис.1.2. task-2.py

Завдання 1.3: Класифікація логістичною регресією або логістичний класифікатор.

Логістична регресія – це методика, що використовується для пояснення відносин між вхідними та вихідними змінними. Вхідні змінні вважаються незалежними, вихідні – залежними. Залежна змінна може мати лише фіксований набір значень.

Лістинг файлу task-3.py

```
import numpy as np
from sklearn import linear_model
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)
```

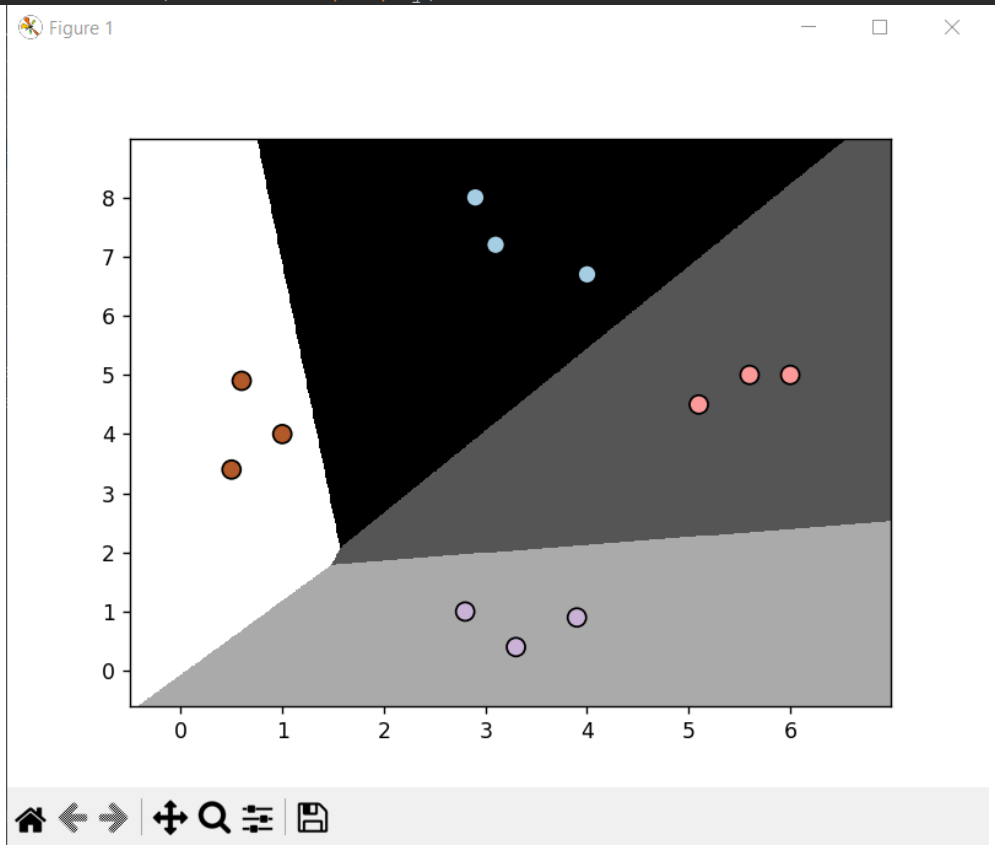

Рис.1.3. task-3.py

Завдання 1.4: Класифікація наївним байєсовським класифікатором. Наївний байєсовський : класифікатор - це простий класифікатор, заснований на використанні теореми Байєса, яка описує ймовірність події з урахуванням пов'язаних з нею умов.

Лістинг файлу task-4.py

```
# 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)

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)) + "%")
```

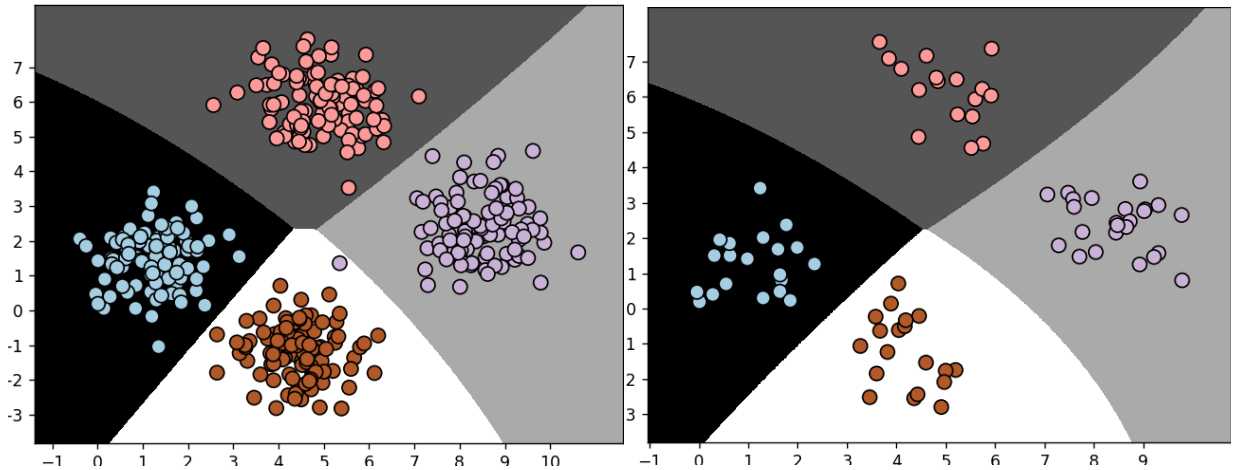

Рис.1.4. task-4.py

Порівняйте між собою результати висновок запишіть у звіт.

Після проведення декількох експериментів, можемо підсумувати, що розділення даних дає можливість отримати більш надійні данні, а використання функції для обчислення якості, точності та повноти даних дає можливість більш детально вказати результат.

Завдання 1.5: Попередня обробка даних: Як правило, при обробці ми маємо справу з великими обсягами

Фрагмент лістингу файлу task-5.py

```
def find_TP(y_true, y_pred):
    # counts the number of true positives (y_true = 1, y_pred = 1)
    return sum((y_true == 1) & (y_pred == 1))

def find_FN(y_true, y_pred):
    # counts the number of false negatives (y_true = 1, y_pred = 0)
    return sum((y_true == 1) & (y_pred == 0))

def find_FP(y_true, y_pred):
    # counts the number of false positives (y_true = 0, y_pred = 1)
    return sum((y_true == 0) & (y_pred == 1))

def find_TN(y_true, y_pred):
    # counts the number of true negatives (y_true = 0, y_pred = 0)
    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
```

```python
# Confusion
def Makovska_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]])

# Accuracy
def Makovska_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)

# Recall


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

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

# F1 score
def Makovska_f1_score(y_true, y_pred):
    precision = Makovska_precision_score(y_true, y_pred)
    recall = Makovska_recal_score(y_true, y_pred)
    return (2 * (precision * recall)) / (precision + recall)
```

```
   actual_label  model_RF  model_LR
0             1  0.639816  0.531904
1             0  0.490993  0.414496
2             1  0.623815  0.569883
3             1  0.506616  0.443674
4             0  0.418302  0.369532
   actual_label  model_RF  model_LR  predicted_RF  predicted_LR
0             1  0.639816  0.531904             1             1
1             0  0.490993  0.414496             0             0
2             1  0.623815  0.569883             1             1
3             1  0.506616  0.443674             1             0
4             0  0.418302  0.369532             0             0
confusion_matrix:
 [[5519 2360]
 [2832 5047]]
TP: 5047
FN: 2832
FP: 2360
TN: 5519
```

```
Makovska_confusion_matrix:
 [[5519 2360]
 [2832 5047]]
Accuracy score on RF: 0.6705165630156111
My accuracy score on RF: 0.6705165630156111
My accuracy score on LR: 0.6158141896179719
Recall score on RF: 0.6405635232897576
My recall score on RF: 0.6405635232897576
My recall score on LR: 0.5430892245208783
Precision score on RF: 0.681382476036182
My precision score on RF: 0.681382476036182
My precision score on LR: 0.6355265112134264
F1 score on RF 0.660342797330891
My F1 score score on RF: 0.660342797330891
My F1 score score on LR: 0.5856830002737475

Scores with threshold = 0.5
Accuracy RF: 0.6705165630156111
Precision RF: 0.681382476036182
Recall RF: 0.6405635232897576
F1 RF: 0.660342797330891
```

```
Scores with threshold = 0.25
Accuracy RF: 0.5024114735372509
Precision RF: 0.5012086513994911
Recall RF: 1.0
F1 RF: 0.6677401584812916

Scores with threshold = 0.6
Accuracy RF: 0.6127681177814444
Precision RF: 0.828952239911144
Recall RF: 0.28417311841604265
F1 RF: 0.42325141776937614

Scores with threshold = 0.2
Accuracy RF: 0.5002538393197106
Precision RF: 0.5001269518852355
Recall RF: 1.0
F1 RF: 0.6667795032369992

AUC RF: 0.73829514083596
AUC LR: 0.6657435203840882
```

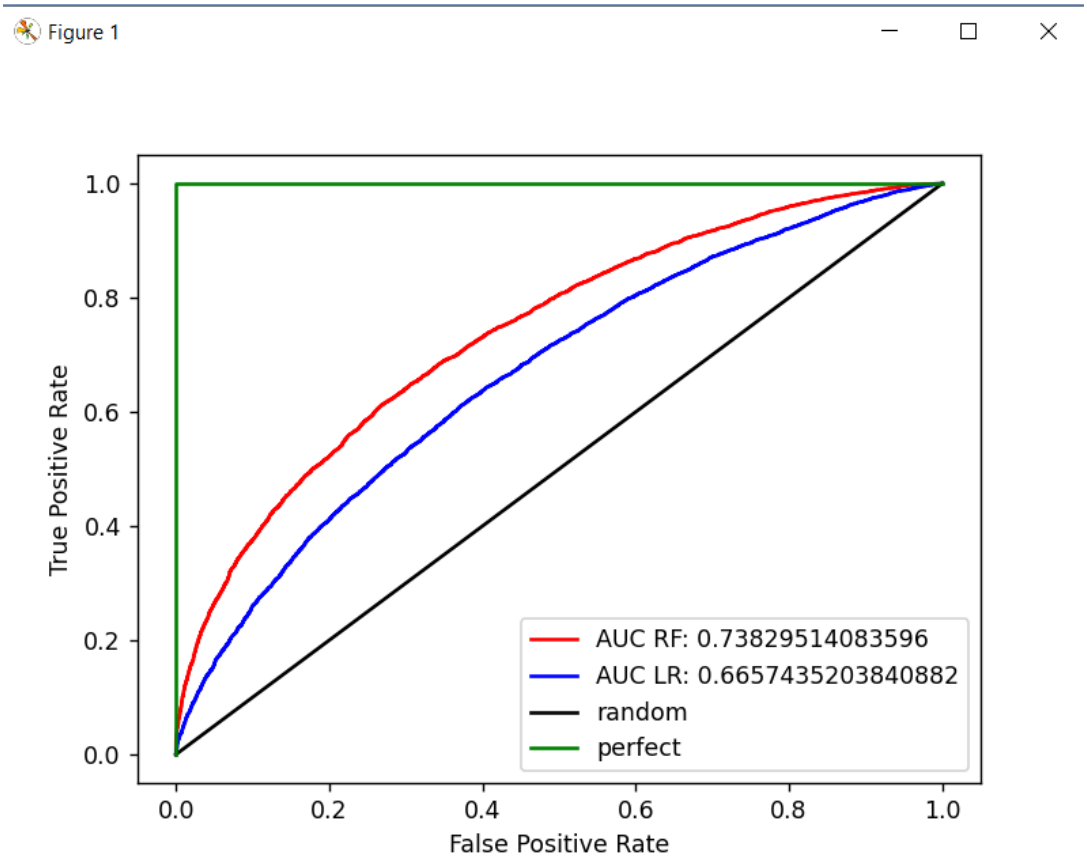

Рис.1.5. task-5.py

Завдання 1.6: Попередня обробка даних: Як правило, при обробці ми маємо справу з великими обсягами

Лістинг файлу task-2.py

```
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=0.21,
random_state=5)
classifier = svm.SVC(decision_function_shape='ovr')
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(), 3)}%")

precision_values = cross_val_score(classifier, X_test, y_test,
scoring='precision_weighted', cv=num_folds)
print(f"Precision: {round(100 * precision_values.mean(), 3)}%")

recall_values = cross_val_score(classifier, X_test, y_test,
scoring='recall_weighted', cv=num_folds)
print(f"Recall: {round(100 * recall_values.mean(), 3)}%")

f1_values = cross_val_score(classifier, X_test, y_test, scoring='f1_weighted',
cv=num_folds)
print(f"F1: {round(100 * f1_values.mean(), 3)}%")
```

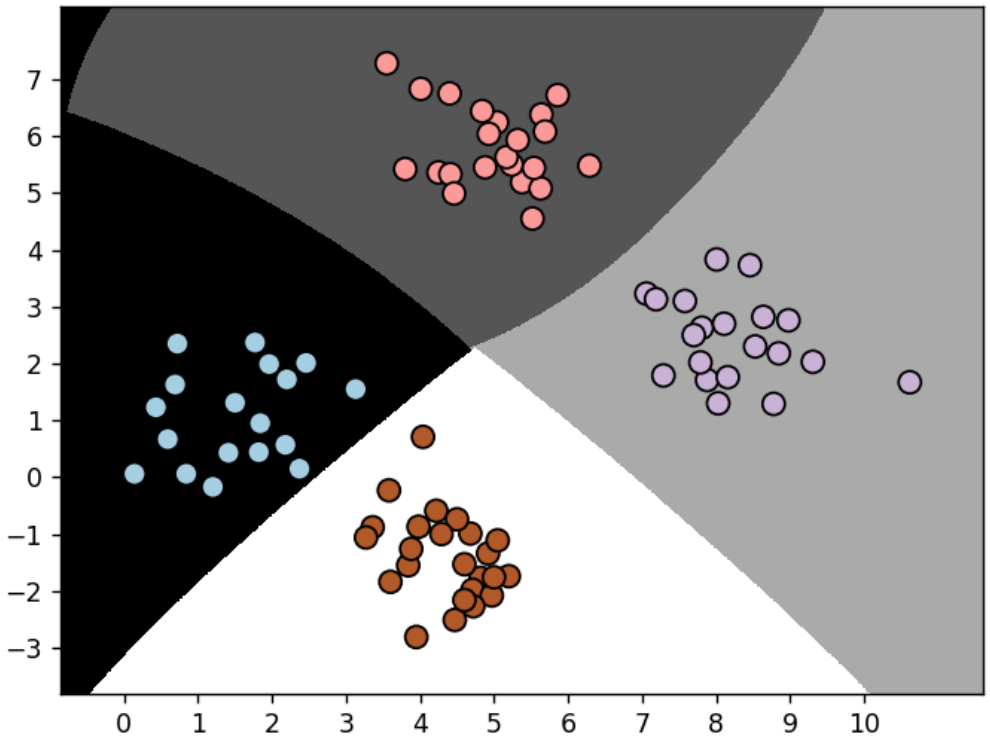

Рис.1.6. task-6.py

https://github.com/avrorilka/AI_Python

Висновки: в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python ми дослідити попередню обробку та класифікацію даних. Розглянули такі методи підготовки та перетворення даних як бінаризація даних, виключення середнього, масштабування, нормалізація, розглянули та проаналізували різницю між L1 та L2 нормалізаціями, та кодування міток. Навчилися застосовувати методику логічної регресії для класифікації логістичною регістрацією та класифікацію наївним байєсовським класифікатором. Також ми вивчили метрики якості класифікації.