**Лабораторна робота №5**

**ДОСЛІДЖЕННЯ АНСАМБЛЕВОГО НАВЧАННЯ**

**Мета роботи:** використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні.

Хід роботи:

**Завдання 5.1** Створення класифікаторів на основі випадкових та гранично випадкових лісів

Лістинг коду:

import argparse   
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier  
from sklearn.metrics import classification\_report  
from utilities import visualize\_classifier  
  
# Argument parser   
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(description='Classify data using \  
 Ensemble Learning techniques')  
 parser.add\_argument('--classifier-type', dest='classifier\_type',   
 required=True, choices=['rf', 'erf'], help="Type of classifier \  
 to use; can be either 'rf' or 'erf'")  
 return parser  
  
if \_\_name\_\_=='\_\_main\_\_':  
 # Parse the input arguments  
 args = build\_arg\_parser().parse\_args()  
 classifier\_type = args.classifier\_type  
  
 # Load input data  
 input\_file = 'data\_random\_forests.txt'  
 data = np.loadtxt(input\_file, delimiter=',')  
 X, y = data[:, :-1], data[:, -1]  
  
 # Separate input data into three classes based on labels  
 class\_0 = np.array(X[y==0])  
 class\_1 = np.array(X[y==1])  
 class\_2 = np.array(X[y==2])

# Visualize input data  
 plt.figure()  
 plt.scatter(class\_0[:, 0], class\_0[:, 1], s=75, facecolors='white',   
 edgecolors='black', linewidth=1, marker='s')

plt.scatter(class\_1[:, 0], class\_1[:, 1], s=75, facecolors='white',   
 edgecolors='black', linewidth=1, marker='o')

plt.scatter(class\_2[:, 0], class\_2[:, 1], s=75, facecolors='white',   
 edgecolors='black', linewidth=1, marker='^')  
plt.title('Input data')  
  
# Split data into training and testing datasets   
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.25, random\_state=5)  
  
# Ensemble Learning classifier  
params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}  
if classifier\_type == 'rf':  
 classifier = RandomForestClassifier(\*\*params)  
else:  
 classifier = ExtraTreesClassifier(\*\*params)  
  
classifier.fit(X\_train, y\_train)  
visualize\_classifier(classifier, X\_train, y\_train)  
  
y\_test\_pred = classifier.predict(X\_test)  
visualize\_classifier(classifier, X\_test, y\_test)  
  
# Evaluate classifier performance  
class\_names = ['Class-0', 'Class-1', 'Class-2']  
print("\n" + "#"\*40)  
print("\nClassifier performance on training dataset\n")  
print(classification\_report(y\_train, classifier.predict(X\_train), target\_names=class\_names))  
print("#"\*40 + "\n")  
  
print("#"\*40)  
print("\nClassifier performance on test dataset\n")  
print(classification\_report(y\_test, y\_test\_pred, target\_names=class\_names))  
print("#"\*40 + "\n")  
  
# Compute confidence  
test\_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]])  
  
print("\nConfidence measure:")  
for datapoint in test\_datapoints:  
 probabilities = classifier.predict\_proba([datapoint])[0]  
 predicted\_class = 'Class-' + str(np.argmax(probabilities))  
 print('\nDatapoint:', datapoint)  
 print('Predicted class:', predicted\_class)   
  
# Visualize the datapoints  
visualize\_classifier(classifier, test\_datapoints, [0]\*len(test\_datapoints))  
  
plt.show()

Результат виконання:

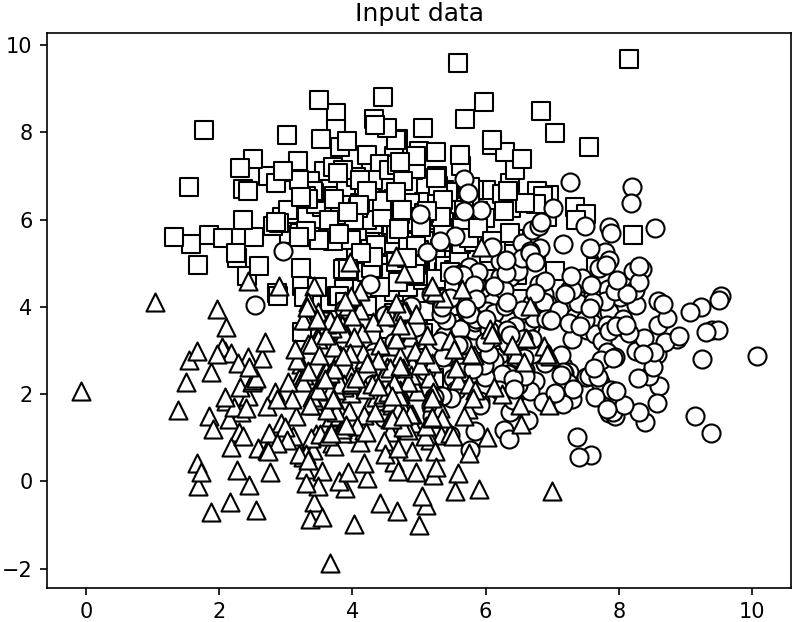


Рисунок 1 - Результат виконання програми (візуалізація введених даних)

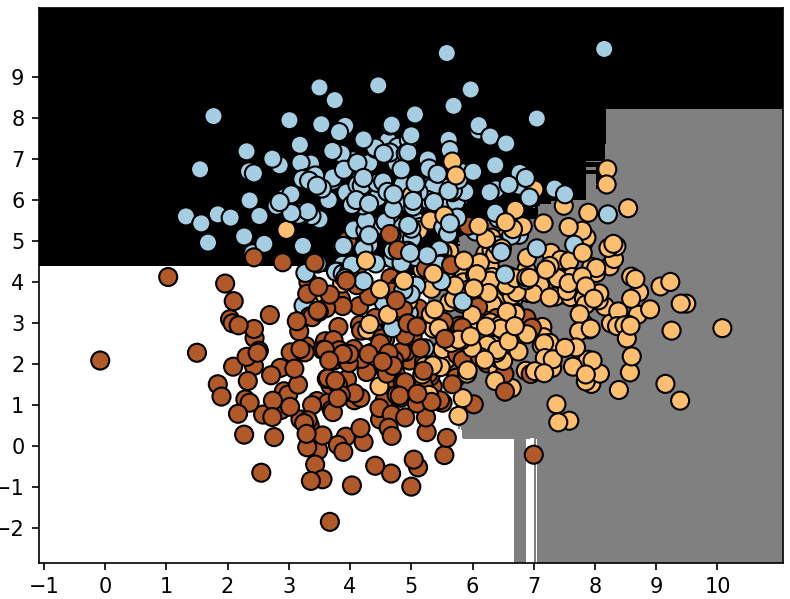


Рисунок 2 - Результат виконання програми (класифікатор на основі випадкового лісу для навчальних даних)

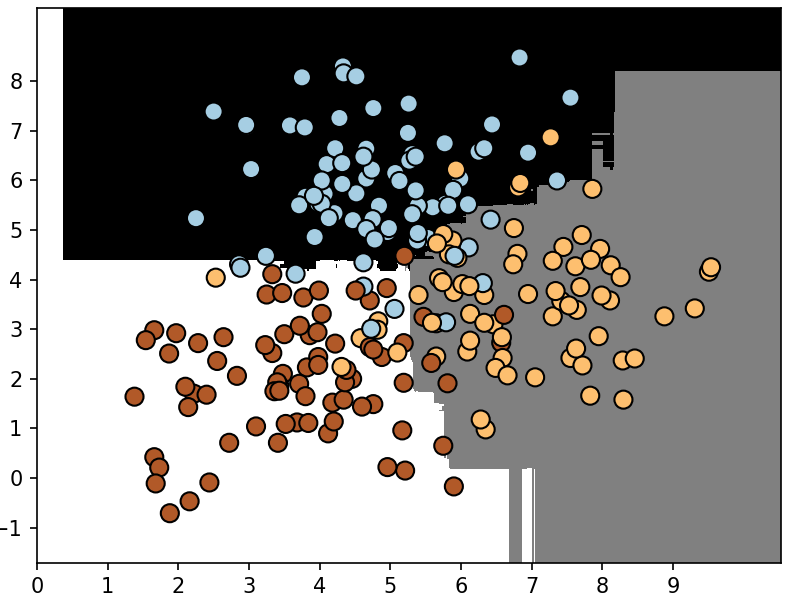


Рисунок 3 - Результат виконання програми (класифікатор на основі випадкового лісу для тестових даних)

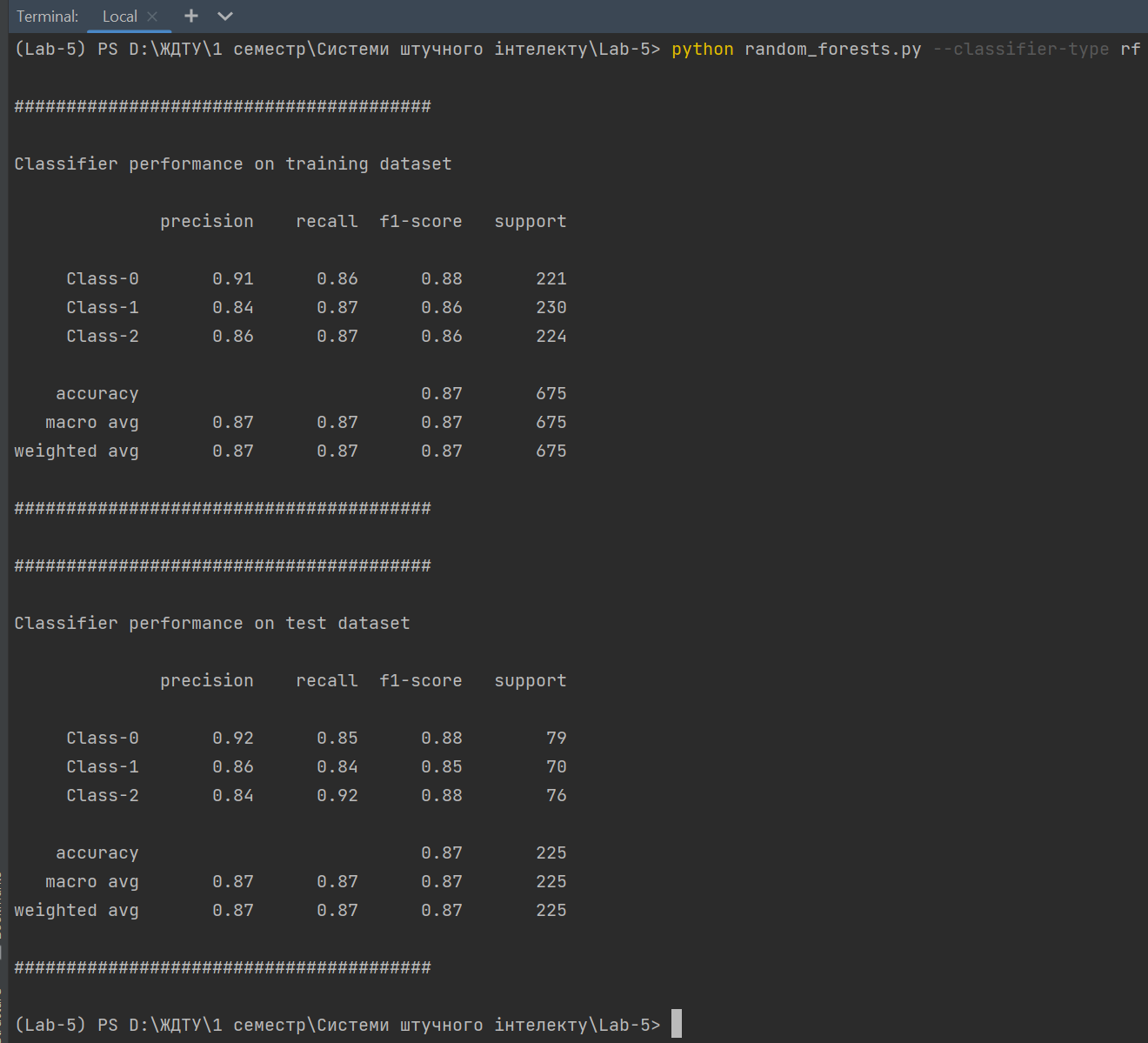


Рисунок 4 - Результат виконання програми

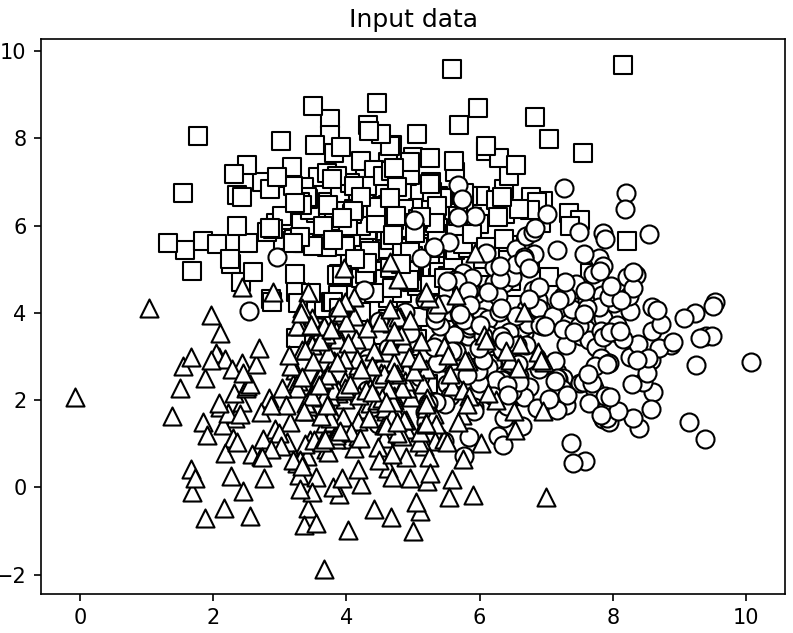


Рисунок 5 - Результат виконання програми (візуалізація введених даних)

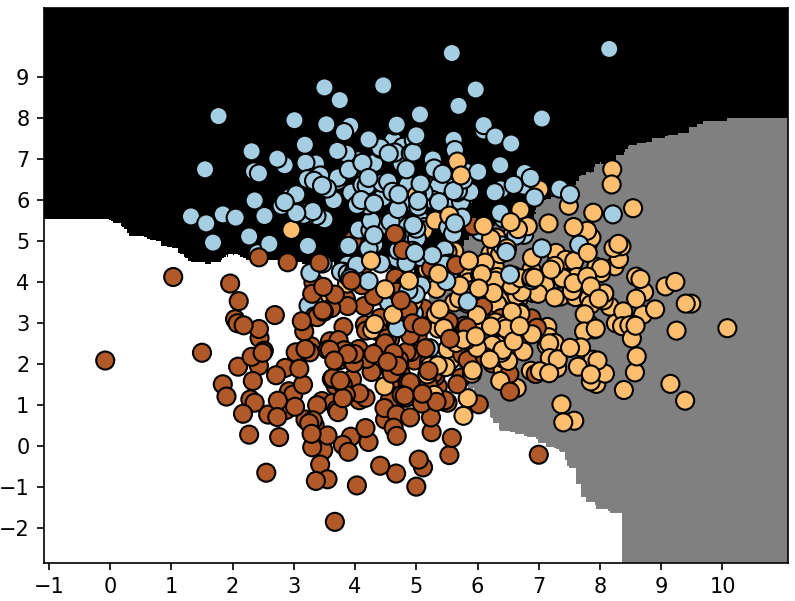


Рисунок 6 - Результат виконання програми (класифікатор на основі граничного випадкового лісу для навчальних даних)

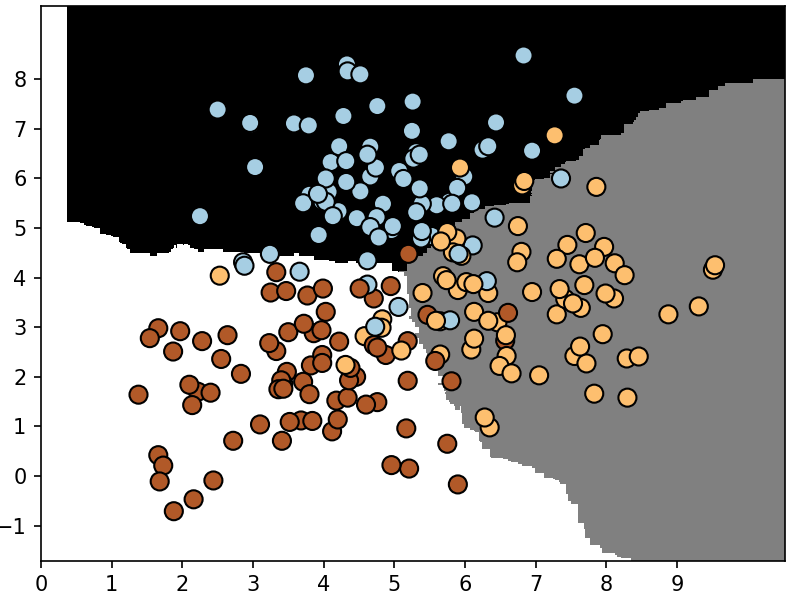


Рисунок 7 - Результат виконання програми (класифікатор на основі граничного випадкового лісу для тестових даних)

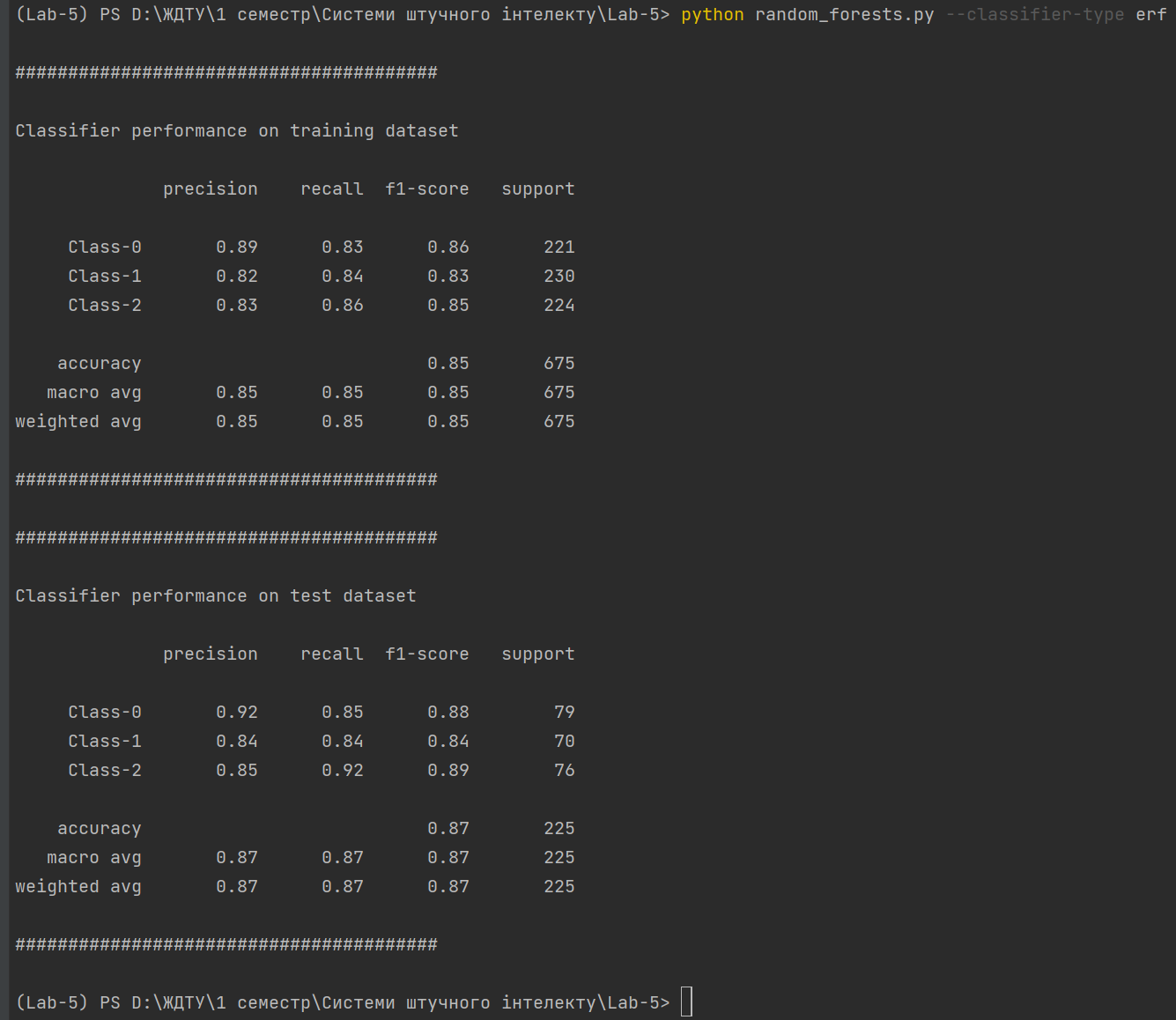


Рисунок 8 - Результат виконання програми

Оцінка мір достовірності прогнозів

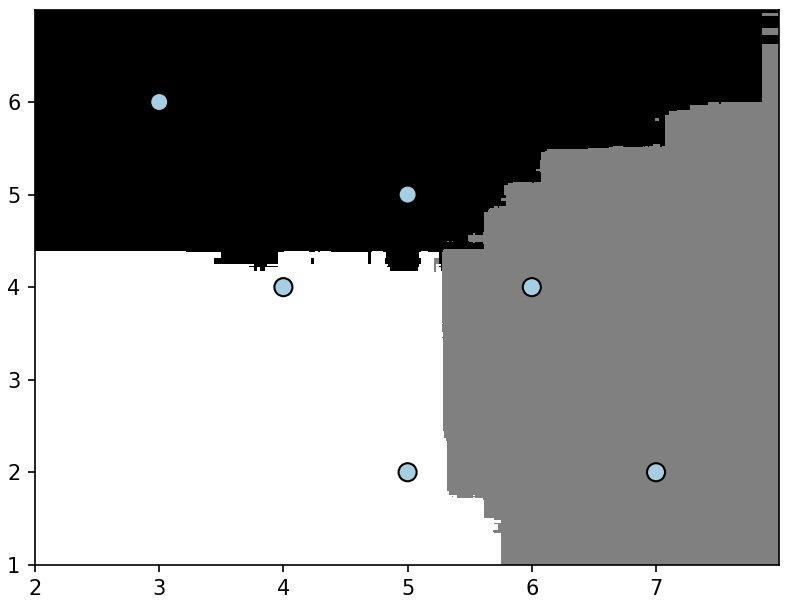


Рисунок 9 - Результат виконання програми на основі випадкового лісу

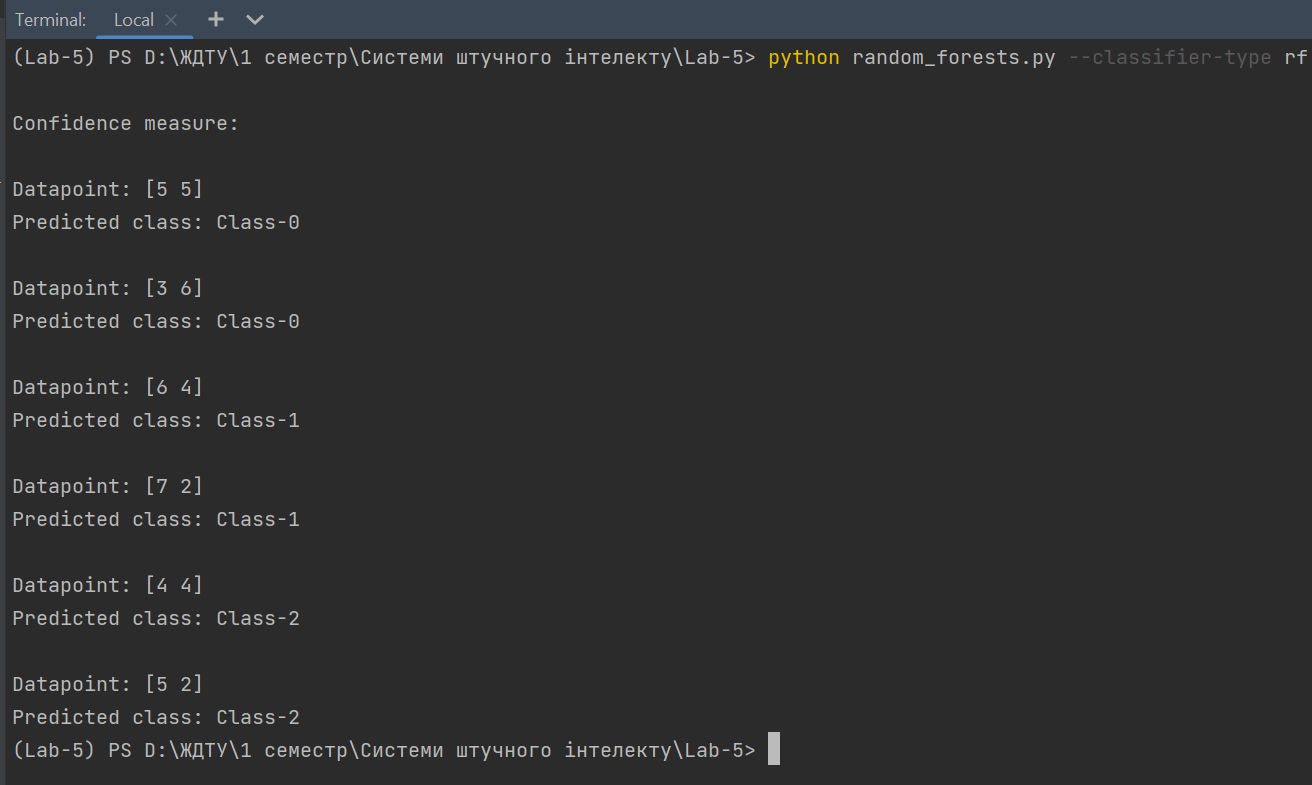


Рисунок 10 - Результат виконання програми на основі випадкового лісу

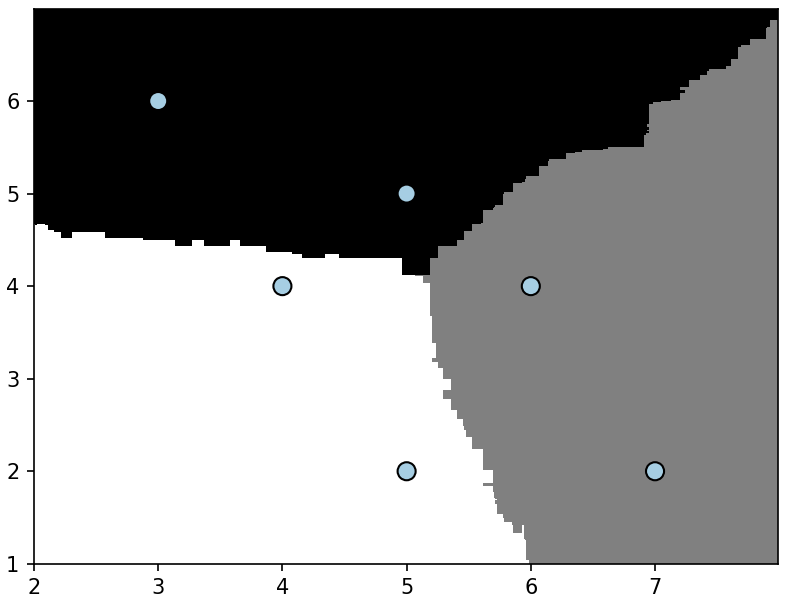


Рисунок 11 - Результат виконання програми на основі граничного випадкового лісу

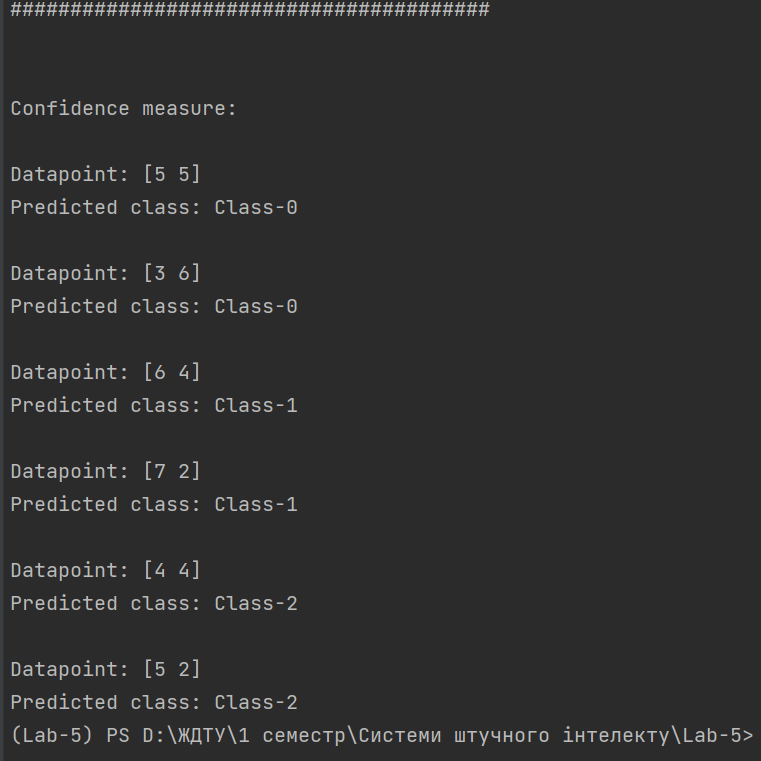


Рисунок 12 - Результат виконання програми на основі граничного випадкового лісу

### **1. На тренувальному наборі даних**

#### Extra Random Forest:

* **Accuracy**: 85%
* **Precision (середнє)**: 85%
* **Recall (середнє)**: 85%
* **F1-score (середнє)**: 85%

#### Random Forest:

* **Accuracy**: 87%
* **Precision (середнє)**: 87%
* **Recall (середнє)**: 87%
* **F1-score (середнє)**: 87%

**Висновок:** Random Forest показує трохи кращі результати на тренувальному наборі, зокрема, на 2% вищу точність.

### **2. На тестовому наборі даних**

#### Extra Random Forest:

* **Accuracy**: 87%
* **Precision (середнє)**: 87%
* **Recall (середнє)**: 87%
* **F1-score (середнє)**: 87%

#### Random Forest:

* **Accuracy**: 87%
* **Precision (середнє)**: 87%
* **Recall (середнє)**: 87%
* **F1-score (середнє)**: 87%

**Висновок:** На тестовому наборі обидва класифікатори показують однакову точність (87%) та інші метрики. Це вказує на те, що обидва моделі мають схожу ефективність у прогнозуванні на нових даних.

### **3. Прогнозування на нових точках (Confidence measure)**

Обидва класифікатори дають однакові прогнози для тестових точок:

* [5, 5] -> Class-0
* [3, 6] -> Class-0
* [6, 4] -> Class-1
* [7, 2] -> Class-1
* [4, 4] -> Class-2
* [5, 2] -> Class-2

### Загальний висновок

1. Random Forest показує трохи кращі результати на тренувальному наборі, але на тестовому наборі метрики однакові для обох класифікаторів.
2. Вибір між Random Forest і Extra Random Forest залежить від задачі:
   * Якщо потрібна краща узгодженість на тренувальних даних **-** Random Forest.
   * Якщо треба уникнути перенавчання через додаткову рандомізацію в розщепленні вузлів, Extra Random Forest може бути більш стійким до перенавчання.

Обидва підходи мають свої переваги, і можна використовувати обидва для порівняння залежно від реальних даних.

**Завдання 5.2** Обробка дисбалансу класів

Лістинг коду:

import sys  
import argparse  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.metrics import classification\_report  
from sklearn.model\_selection import train\_test\_split  
from utilities import visualize\_classifier  
  
# Завантаження вхідних даних  
input\_file = 'data\_imbalance.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
# Поділ вхідних даних на два класи на підставі міток  
class\_0 = np.array(X[y == 0])  
class\_1 = np.array(X[y == 1])  
  
# Візуалізація вхідних даних  
plt.figure()  
plt.scatter(class\_0[:, 0], class\_0[:, 1], s=75, facecolors='black', edgecolors='black', linewidth=1, marker='x')  
plt.scatter(class\_1[:, 0], class\_1[:, 1], s=75, facecolors='white', edgecolors='black', linewidth=1, marker='o')  
plt.title('Input data')  
  
# Розбиття даних на навчальний та тестовий набори  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.25, random\_state=5)  
  
# Класифікатор на основі гранично випадкових лісів  
params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}  
if len(sys.argv) > 1:  
 if sys.argv[1] == 'balance':  
 params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0, 'class\_weight': 'balanced'}  
 else:  
 raise TypeError("Invalid input argument; should be 'balance'")  
  
classifier = ExtraTreesClassifier(\*\*params)  
classifier.fit(X\_train, y\_train)  
visualize\_classifier(classifier, X\_train, y\_train)  
  
# Передбачимо та візуалізуємо результат для тестового набору даних  
y\_test\_pred = classifier.predict(X\_test)  
visualize\_classifier(classifier, X\_test, y\_test)  
  
# Обчислення показників ефективності класифікатора  
class\_names = ['Class-0', 'Class-1']  
print("\n" + "#" \* 40)  
print("\nClassifier performance on training dataset\n")  
print(classification\_report(y\_train, classifier.predict(X\_train), target\_names=class\_names))  
print("#" \* 40 + "\n")  
  
print("#" \* 40)  
print("\nClassifier performance on test dataset\n")  
print(classification\_report(y\_test, y\_test\_pred, target\_names=class\_names))  
print("#" \* 40 + "\n")  
plt.show()

**Результат виконання програми:**

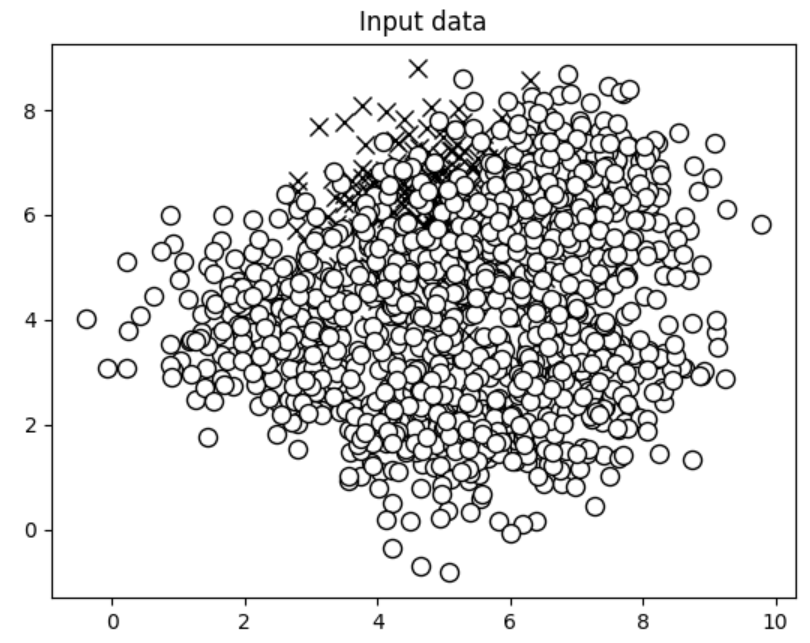


Рисунок 13 – Графік вхідних даних

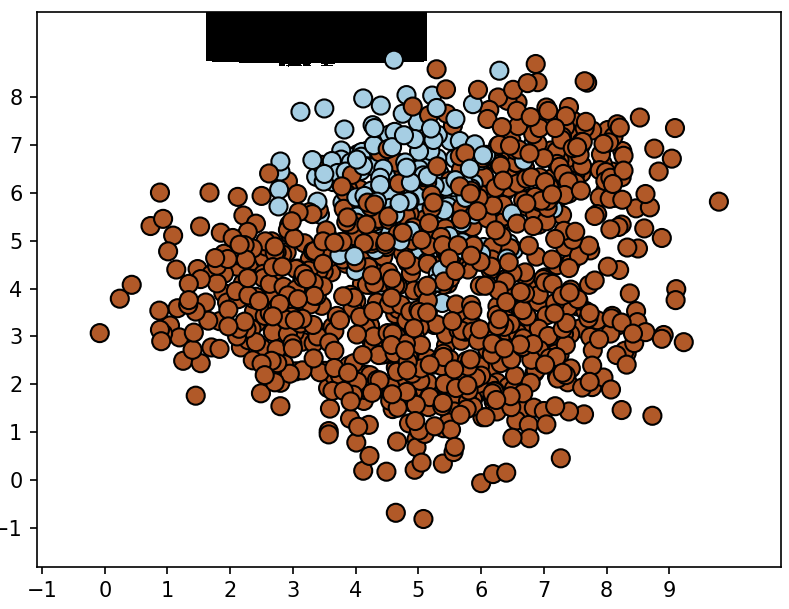


Рисунок 14 – Графік даних класифікатора для навчального набору

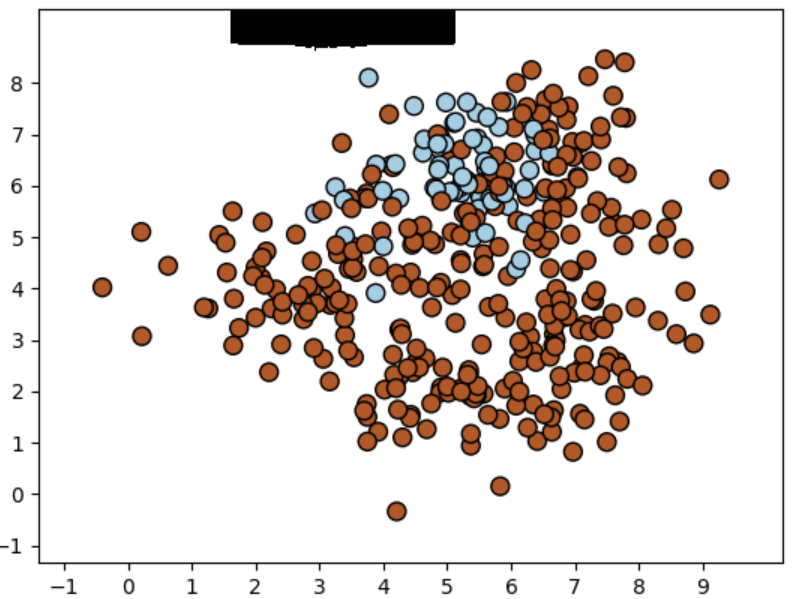


Рисунок 15 – Графік даних класифікатора для тестового набору

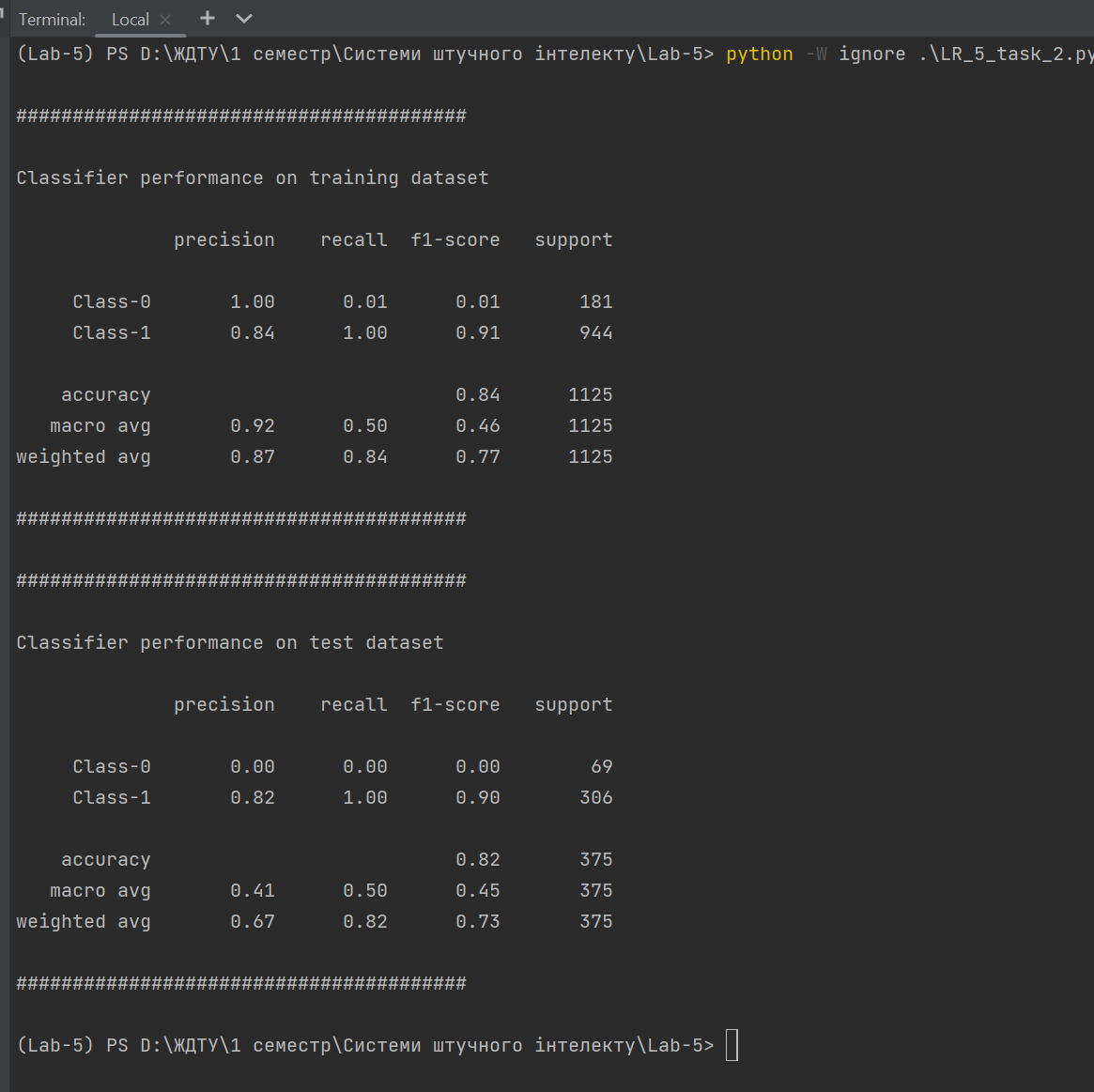


Рисунок 16 – Результат виконання програми

На основі результатів класифікації з використанням моделі випадкових лісів за умов дисбалансу класів, можна зробити такі висновки:

1. **Результати на тренувальній вибірці**

* **Class-0 (менш представлений клас):**
* **Precision:** 1.00, але **Recall:** лише 0.01. Це вказує на те, що модель майже не знаходить прикладів цього класу.
* **F1-score:** 0.01 — дуже низький показник.
* **Class-1 (переважний клас):**
* **Precision:** 0.84, а **Recall:** 1.00. Модель добре знаходить приклади класу, але може включати хибнопозитивні значення.
* **F1-score:** 0.91 — добрий результат.
* Загальна **accuracy:** 0.84, але:
* **Macro avg F1-score:** 0.46 — значно нижчий, що вказує на суттєвий дисбаланс у продуктивності для різних класів.
* **Weighted avg F1-score:** 0.77 — відображає перекіс у бік Class-1.

1. **Результати на тестовій вибірці**

* **Class-0:**
  + Precision, Recall, F1-score — 0.00. Модель не змогла розпізнати цей клас на тестовій вибірці.
* **Class-1:**
  + **Precision:** 0.82, **Recall:** 1.00, **F1-score:** 0.90. Модель майже ідеально працює з переважним класом.
* Загальна **accuracy:** 0.82, але:
  + **Macro avg F1-score:** 0.45 — дуже низький, через провал у класифікації Class-0.
  + **Weighted avg F1-score:** 0.73 — прийнятний рівень, але знову ж таки через перекіс

**Завдання 5.3** Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

Лістинг коду:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import classification\_report  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.metrics import classification\_report  
from utilities import visualize\_classifier  
  
# Завантаження вхідних даних  
input\_file = 'data\_random\_forests.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
# Розбиття даних на три класи на підставі міток  
class\_0 = np.array(X[y == 0])  
class\_1 = np.array(X[y == 1])  
class\_2 = np.array(X[y == 2])  
  
# Розбиття даних на навчальний та тестовий набори  
X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.25, random\_state=5)  
  
# Визначення сітки значень параметрів  
parameter\_grid = [{'n\_estimators': [100], 'max\_depth': [2, 4, 7, 12, 16]},  
 {'max\_depth': [4], 'n\_estimators': [25, 50, 100, 250]}]  
  
#Визначаємо метричні характеристики  
metrics = ['precision\_weighted', 'recall\_weighted']  
  
#Сітковий пошук  
for metric in metrics:  
 print("\n##### Searching optimal parameters for", metric)  
  
 classifier = GridSearchCV(  
 ExtraTreesClassifier(random\_state=0),  
 parameter\_grid, cv=5, scoring=metric)  
 classifier.fit(X\_train, y\_train)  
  
 # Вивід оцінки для кожної комбінації параметрів  
 print("\nGrid scores for the parameter grid:")  
 for i in range(len(classifier.cv\_results\_['params'])):  
 print(  
 classifier.cv\_results\_['params'][i],  
 '-->',  
 round(classifier.cv\_results\_['mean\_test\_score'][i], 3)  
 )  
  
 #Результати роботи класифікатора  
 y\_pred = classifier.predict(X\_test)  
 print("\nPerformance report:\n")  
 print(classification\_report(y\_test, y\_pred))

**Результат виконання програми:**

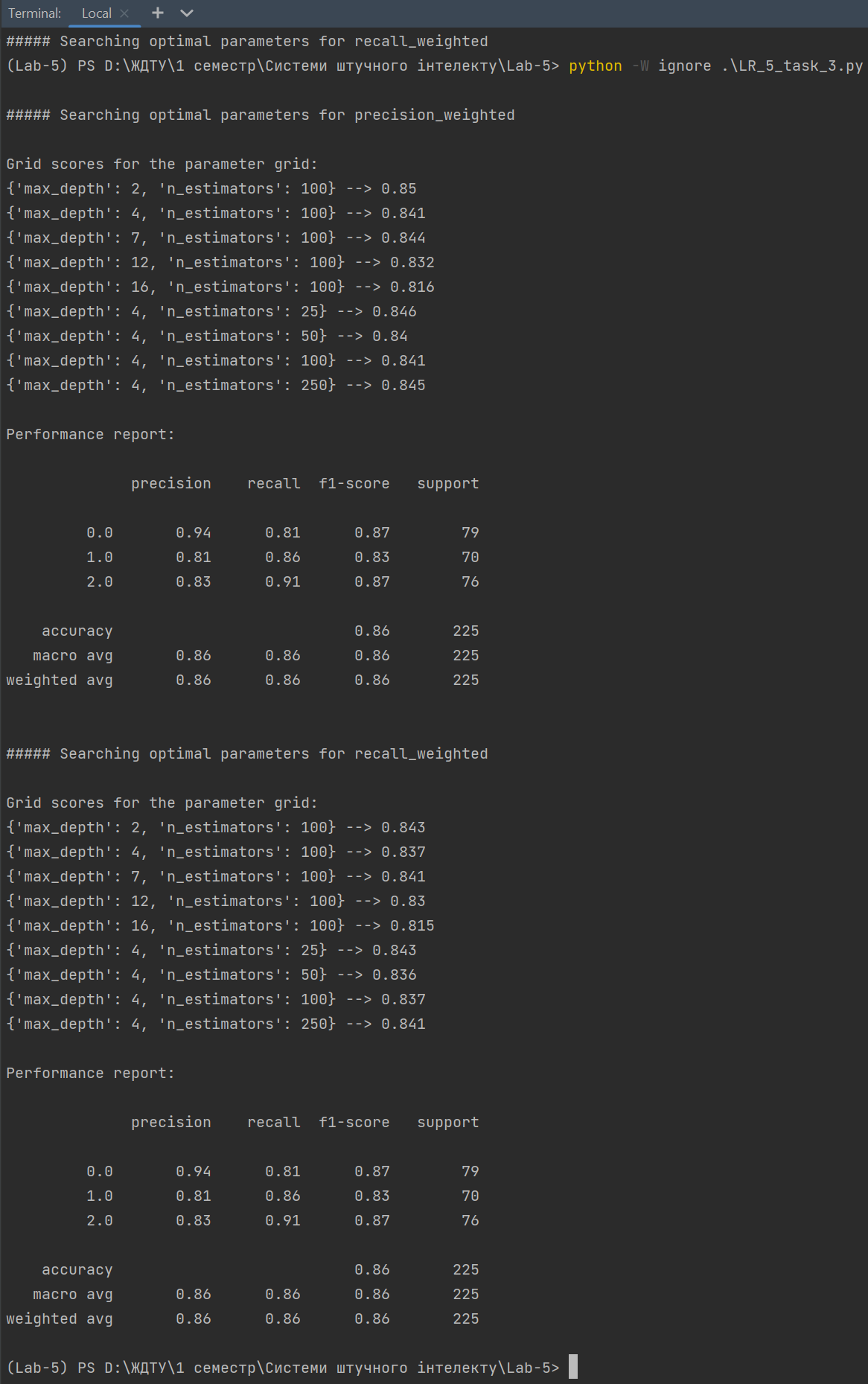


Рисунок 17 – Результат виконання програми

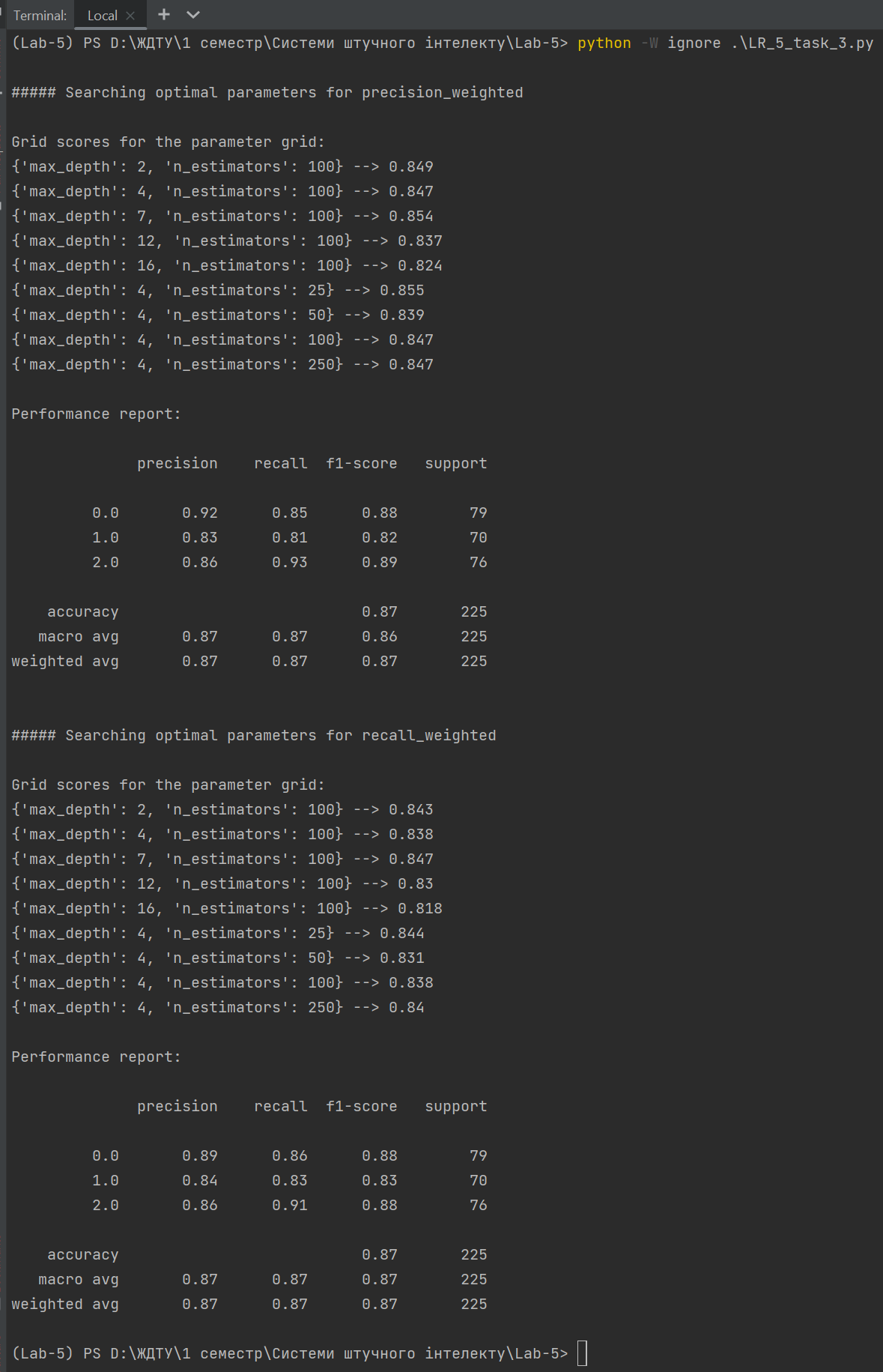


Рисунок 18 – Результат виконання програми (після зміни параметрів)

1. **Результати для метрики precision\_weighted:**

* Найкраща комбінація параметрів: max\_depth = 2 і n\_estimators = 100, з оцінкою 0.85.
* В цілому, параметри з max\_depth = 2 та різними значеннями n\_estimators (особливо 25, 50 і 100) показали найкращі результати для цієї метрики.
* Результати класифікації показують високі показники точності та відсоткові оцінки для кожного класу, з середньою точністю (precision) 0.86.

1. **Результати для метрики recall\_weighted:**

* Найкраща комбінація параметрів: max\_depth = 2 і n\_estimators = 100, з оцінкою 0.843.
* Результати для recall показують схожі тенденції, де комбінація з max\_depth = 2 і різними значеннями n\_estimators також дає кращі результати, хоча з трохи нижчими значеннями, порівняно з precision.
* Загальний результат для recall схожий: висока точність та схоже розподілення по класах, з середнім recall 0.86.

1. Основні спостереження:

* Обидві метрики (precision і recall) мають схожі результати, з найкращими комбінаціями параметрів, що включають max\_depth = 2 і n\_estimators = 100.
* Незважаючи на відмінності в метриках, модель показала хороші загальні результати для всіх класів з високою точністю і recall.

Після зміни параметрів **random\_state** та **cv** деякі показники для сітки покращились, а деякі погіршились.

**Завдання 5.4** Обчислення відносної важливості ознак

Лістинг коду:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.tree import DecisionTreeRegressor  
from sklearn.ensemble import AdaBoostRegressor  
from sklearn import datasets  
from sklearn.metrics import mean\_squared\_error, explained\_variance\_score  
from sklearn.utils import shuffle  
from sklearn.datasets import fetch\_california\_housing  
from sklearn.model\_selection import train\_test\_split  
  
# datasets.load\_boston() Вилучино в нових версіях  
# Завантаження даних із цінами на нерухомість  
housing\_data = datasets.fetch\_california\_housing()  
  
# Перемішування даних  
X, y = shuffle(housing\_data.data, housing\_data.target, random\_state=7)  
  
# Розбиття 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=7)

# Модель на основі регресора AdaBoost  
regressor = AdaBoostRegressor(DecisionTreeRegressor(max\_depth=4),  
 n\_estimators=400, random\_state=7)  
regressor.fit(X\_train, y\_train)  
  
# Обчислення показників ефективності регресора AdaBoost  
y\_pred = regressor.predict(X\_test)  
mse = mean\_squared\_error(y\_test, y\_pred)  
evs = explained\_variance\_score(y\_test, y\_pred)  
print("\nADABOOST REGRESSOR")  
print("Mean squared error =", round(mse, 2))  
print("Explained variance score =", round(evs, 2))  
  
# Вилучення важливості ознак  
feature\_importances = regressor.feature\_importances\_  
feature\_names = np.array(housing\_data.feature\_names)  
  
# Нормалізація значень важливості ознак  
feature\_importances = 100.0 \* (feature\_importances / max(feature\_importances))  
  
# Сортування та перестановка значень  
index\_sorted = np.flipud(np.argsort(feature\_importances))  
  
# Розміщення міток уздовж осі Х  
pos = np.arange(index\_sorted.shape[0]) + 0.5  
  
# Побудова стовпчастої діаграми (модернізована, бо по осі х погано відображалося)  
plt.figure(figsize=(10, 6)) # Збільшення розміру фігури  
plt.bar(pos, feature\_importances[index\_sorted], align='center')  
plt.xticks(pos, feature\_names[index\_sorted], rotation=45, ha='right') # Обертання міток  
plt.ylabel('Relative Importance')  
plt.title('Feature importance using AdaBoost regressor')  
plt.tight\_layout() # Оптимізація розташування компонентів  
plt.show()

**Результат виконання програми:**

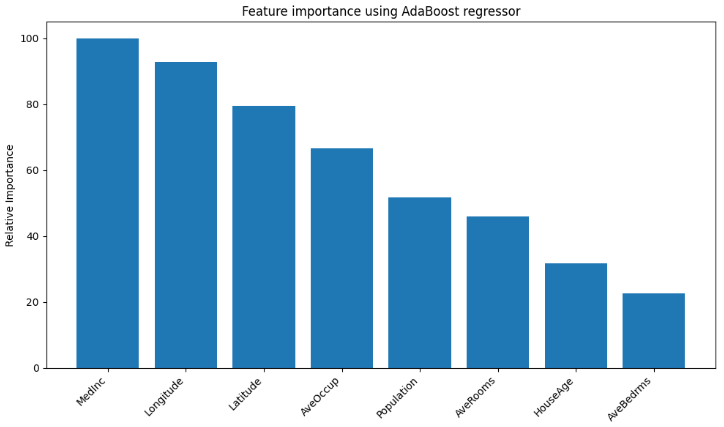


Рисунок 19 – Результат виконання програми

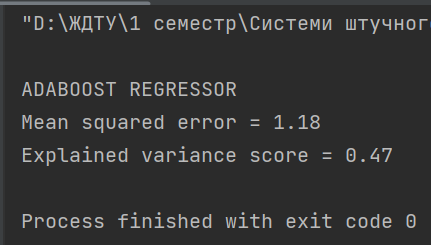


Рисунок 20 – Результат виконання програми

На основі проведеного аналізу, найважливішою ознакою для прогнозування цін на нерухомість є медіанний дохід (*MedInc*), оскільки його вплив найбільший серед усіх факторів. Це свідчить про сильну залежність вартості нерухомості від рівня доходів населення в регіоні. Географічне розташування, представлене довготою (*Longitude*) та широтою (*Latitude*), також має вагомий вплив, підтверджуючи важливість локації для визначення цін. Крім того, середня кількість мешканців у будинку (*AveOccup*) є суттєвим фактором, що може бути пов’язано із соціально-демографічними особливостями районів.

Середні значення для населення району (*Population*) та кількості кімнат у будинку (*AveRooms*) мають помірний вплив, що вказує на їх певну, але не критичну роль у формуванні цін. Менш важливими виявилися вік будинку (*HouseAge*) та середня кількість спалень (*AveBedrms*), що свідчить про їхній обмежений вплив на кінцевий результат. Таким чином, для побудови більш оптимізованої моделі варто зосередитися на ключових ознаках, таких як *MedInc*, *Longitude*, *Latitude* та *AveOccup*, тоді як менш значимі фактори можна враховувати опціонально або вилучати для спрощення аналізу.

**Завдання 5.5** Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів

Лістинг коду:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import classification\_report, mean\_absolute\_error  
from sklearn import preprocessing  
from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import ExtraTreesRegressor  
from sklearn.metrics import classification\_report  
  
# Завантаження вхідних даних  
input\_file = 'traffic\_data.txt'  
data = []  
with open(input\_file, 'r') as f:  
 for line in f.readlines():  
 items = line[:-1].split(',')  
 data.append(items)  
  
data = np.array(data)  
  
# Перетворення рядкових даних на числові  
label\_encoder = []  
X\_encoded = np.empty(data.shape)  
for i, item in enumerate(data[0]):  
 if item.isdigit():  
 X\_encoded[:, i] = data[:, i]  
 else:  
 label\_encoder.append(preprocessing.LabelEncoder())  
 X\_encoded[:, i] = label\_encoder[-1].fit\_transform(data[:, i])  
  
X = X\_encoded[:, :-1].astype(int)  
y = X\_encoded[:, -1].astype(int)  
  
# Розбиття даних на навчальний та тестовий набори  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.25, random\_state=5)  
  
# Регресор на основі гранично випадкових лісів  
params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}  
regressor = ExtraTreesRegressor(\*\*params)  
regressor.fit(X\_train, y\_train)  
  
# Обчислення характеристик ефективності регресора на тестових даних  
y\_pred = regressor.predict(X\_test)  
print("Mean absolute error:", round(mean\_absolute\_error(y\_test, y\_pred), 2))  
  
# Тестування кодування на одиночному прикладі  
test\_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']  
test\_datapoint\_encoded = [-1] \* len(test\_datapoint)  
count = 0  
for i, item in enumerate(test\_datapoint):  
 if item.isdigit():  
 test\_datapoint\_encoded[i] = int(test\_datapoint[i])  
 else:  
 test\_datapoint\_encoded[i] = int(label\_encoder[count].transform([test\_datapoint[i]])[0])  
 count = count + 1  
test\_datapoint\_encoded = np.array(test\_datapoint\_encoded)  
  
# Прогнозування результату для тестової точки даних  
print("Predicted traffic:", int(regressor.predict([test\_datapoint\_encoded])[0]))

**Результат виконання програми:**

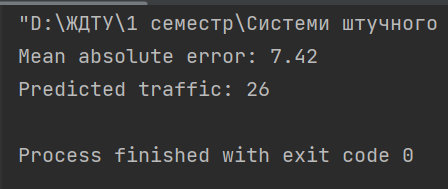


Рисунок 21 – Результат виконання програми

**Висновок:** використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні.