**ЛАБОРАТОРНА РОБОТА №4**

**ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА**

**СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ**

**Мета**: використовуючи спеціалізовані бібліотеки і мову програмування Python дослідити методи ансамблів машинному навчанні.

**Хід роботи:**

**Завдання 1: Створення класифікаторів на основі випадкових та гранично випадкових лісів.**

Використовувати файл вхідних даних: data\_random\_forests.txt, побудувати класифікатори на основі випадкових та гранично випадкових лісів.

Лістинг програми:

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  
  
  
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\_\_":  
 args = build\_arg\_parser().parse\_args()  
 classifier\_type = args.classifier\_type  
  
 input\_file = "data\_random\_forests.txt"  
 data = np.loadtxt(input\_file, delimiter=",")  
 X, y = data[:, :-1], data[:, -1]  
  
 class\_0 = np.array(X[y == 0])  
 class\_1 = np.array(X[y == 1])  
 class\_2 = np.array(X[y == 2])

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")  
  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.25, random\_state=5  
 )  
  
 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)  
  
 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")  
  
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\_classifier(classifier, test\_datapoints, [0] \* len(test\_datapoints))  
plt.show()

Результат виконання за допомогою створення класифікатора на основі випадкового лісу, прапорець rf вхідного аргументу:

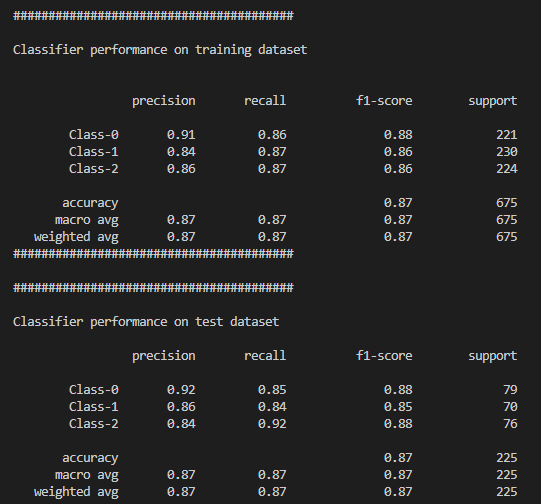
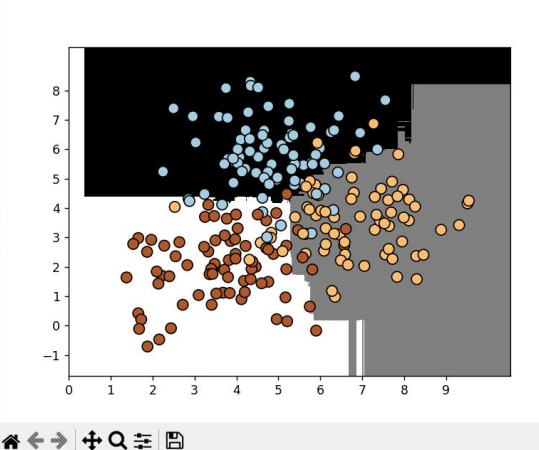
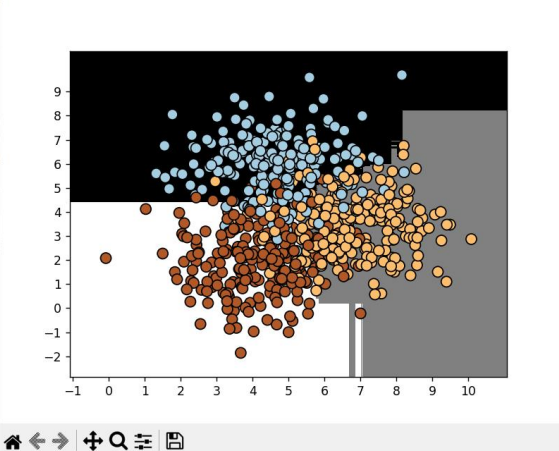
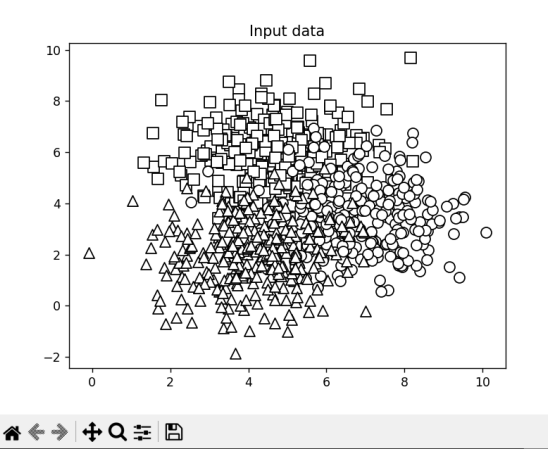
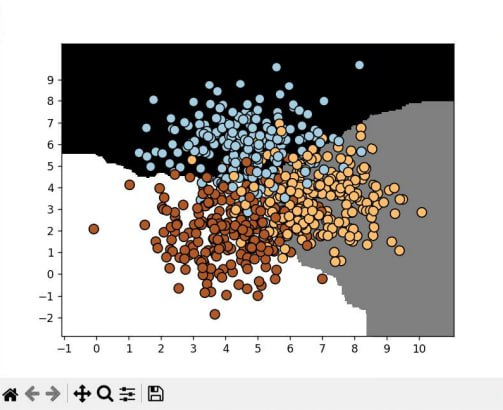
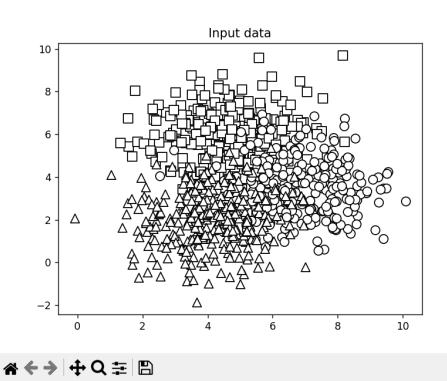


Рис. 1.1 – 1.4. Результат виконання (rf)

Результат виконання за допомогою створення класифікатора на основі гранично випадкового лісу, прапорець erf вхідного аргументу:



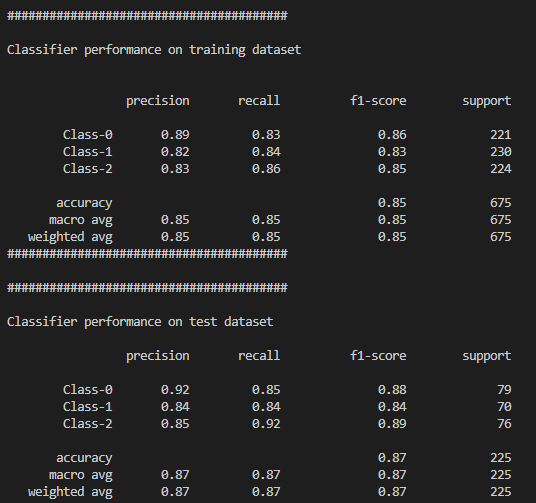
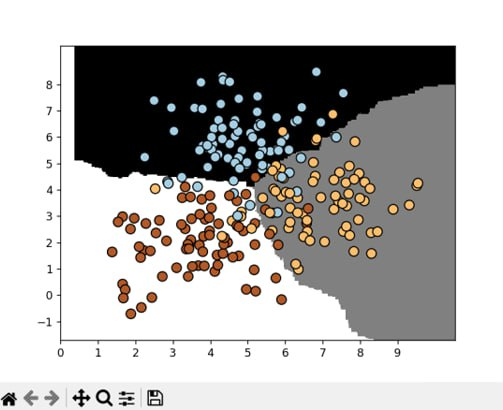


Рис. 1.5 – 1.8. Результат виконання (erf)

Оцінка мір достовірності прогнозів

Якщо ви подивитеся на результати, що відображаються у вікні терміналу, побачите, що для кожної точки даних виводяться ймовірності. Цими ймовірностями вимірюються рівні довірливості (рівні довіри) для кожного класу. Оцінка рівнів довіри відіграє важливу роль у машинному навчанні.

Лістинг програми (доповнення):

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\_classifier(classifier, test\_datapoints, [0] \* len(test\_datapoints))  
plt.show()

Результат виконання за допомогою створення класифікатора на основі випадкового лісу, прапорець rf вхідного аргументу:

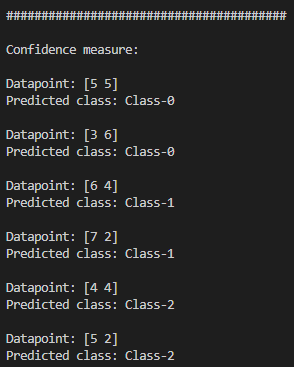
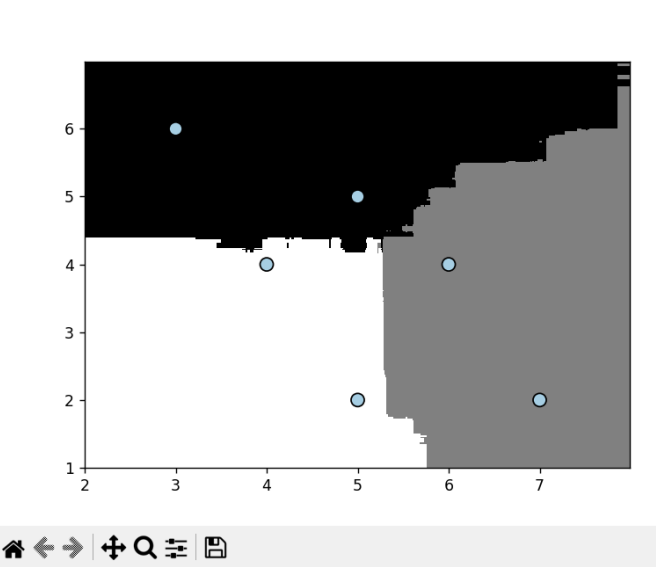


Рис. 1.9 – 1.10. Результат виконнаня (доповнення, rf)

Результат виконання за допомогою створення класифікатора на основі

випадкового лісу, прапорець erf вхідного аргументу:

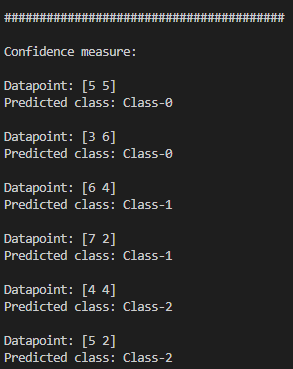
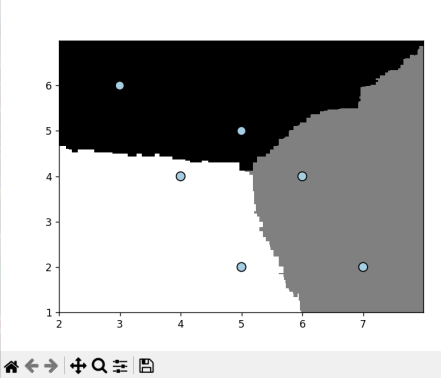


Рис. 1.10 – 1.11. Результат виконання (доповнення, erf)

Зробіть висновок

Обидва методи класифікації (Random Forest та Extra Trees) виявилися ефективними у вирішенні даної задачі класифікації даних. Вони показали приблизно однакову точність, відновлення та F1-показники як на навчальному, так і на тестовому наборі даних. Це свідчить про їхню здатність до узагальнення та стійкість до перенавчання. Результати параметрів довірливості також додатково підтверджують надійність цих класифікаторів для вирішення задачі класифікації даних.

**Завдання 2: Обробка дисбалансу класів**.

Використовуючи для аналізу дані, які містяться у файлі data\_imbalance.txt проведіть обробку з урахуванням дисбалансу класів.

Лістинг програми:

import sys  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report  
from utilities import visualize\_classifier  
  
input\_file = "data\_imbalance.txt"  
data = np.loadtxt(input\_file, delimiter=",")  
X, y = data[:, :-1], data[:, -1]  
  
class\_0 = np.array(X[y == 0])  
class\_1 = np.array(X[y == 1])  
  
plt.figure()  
plt.scatter(class\_0[:, 0], class\_0[:, 1], s=75, color="black", marker="x")  
plt.scatter(  
 class\_1[:, 0],  
 class\_1[:, 1],  
 s=75,  
 facecolors="white",  
 edgecolors="black",  
 linewidth=1,  
 marker="o",  
)  
plt.title("Input data")  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.25, random\_state=5  
)  
  
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)  
y\_test\_pred = classifier.predict(X\_test)  
visualize\_classifier(classifier, X\_test, y\_test)  
  
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, zero\_division=1  
 )  
)  
print("#" \* 40 + "\n")  
print("#" \* 40)  
print("\nClassifier performance on test dataset\n")  
print(  
 classification\_report(  
 y\_test, y\_test\_pred, target\_names=class\_names, zero\_division=1  
 )  
)  
print("#" \* 40 + "\n")  
plt.show()

Результат виконання для незбалансованих даних:

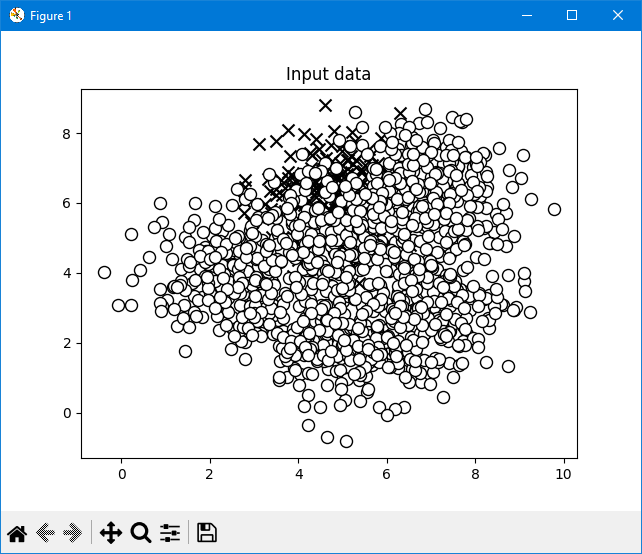


Рис. 2.1. Графік вхідних даних (незбалансованих)

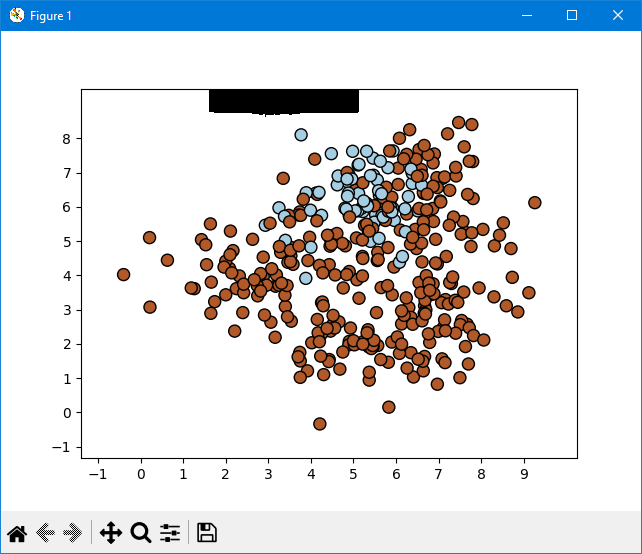
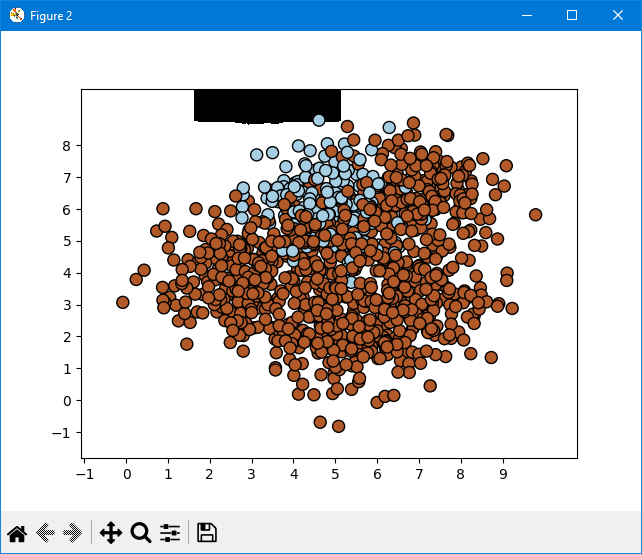


Рис. 2.2 – 2.3. Графіки даних класифікатора для навчального та тестового наборів

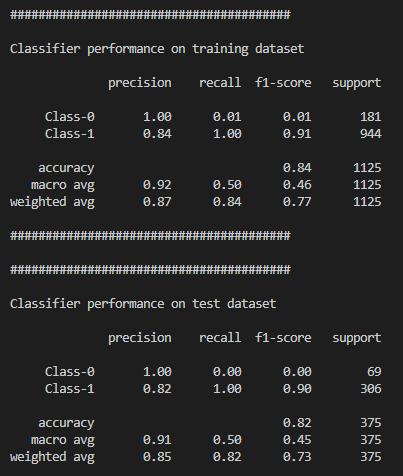


Рис. 2.4. Скріншот з інформацією з терміналу

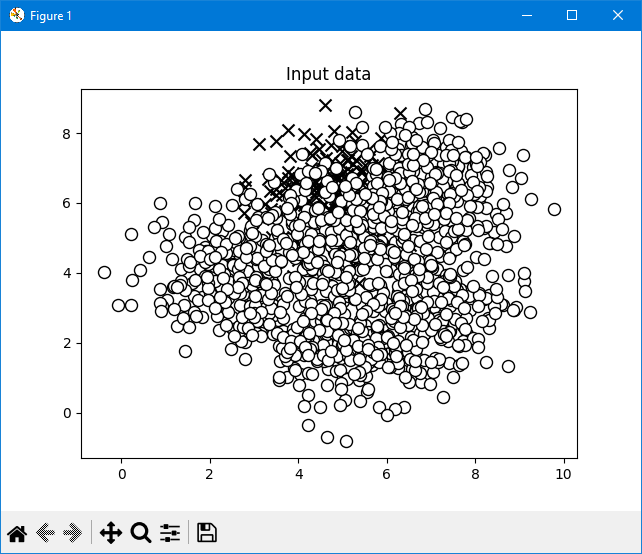


Рис. 2.5. Графік вхідних даних (збалансованих)

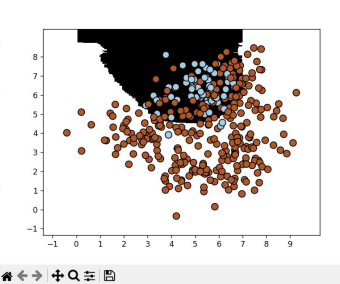
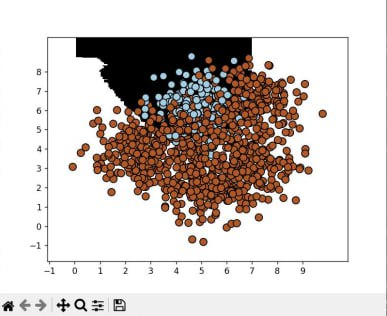


Рис. 2.6 – 2.7. Графіки даних класифікатора для навчального та тестового наборів

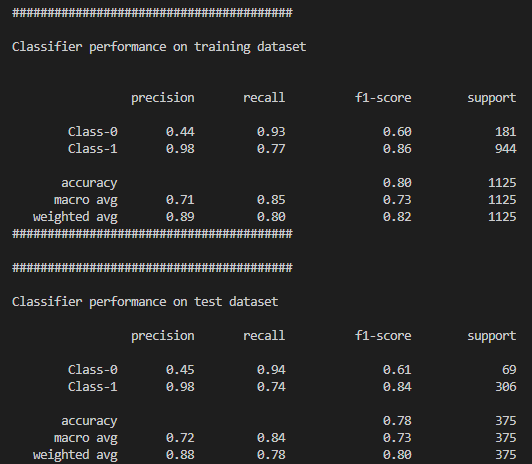


Рис. 2.8. Скріншот з інформацією з терміналу

Зробіть висновок

Використання збалансованих даних (з параметром class\_weight='balanced') призвело до значного поліпшення результатів класифікації, особливо для меншого класу (Class-0). Точність, recall і f1-score для Class-0 значно зросли, і метрики стали більш збалансованими для обох класів. Це свідчить про важливість обробки дисбалансу даних у задачах класифікації. Збалансовані дані допомагають класифікатору краще розпізнавати менший клас і, в результаті, покращують загальну ефективність моделі. Таким чином, при роботі з незбалансованими даними рекомендується розглядати методи обробки дисбалансу для досягнення кращих результатів класифікації.

**Завдання 3: Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку.**

Використовуючи дані, що містяться у файлі data\_random\_forest.txt знайти оптимальних навчальних параметрів за допомогою сіткового пошуку.

Лістинг програми:

import numpy as np  
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  
  
input\_file = "data\_random\_forests.txt"  
data = np.loadtxt(input\_file, delimiter=",")  
X, y = data[:, :-1], data[:, -1]  
  
class\_0 = np.array(X[y == 0])  
class\_1 = np.array(X[y == 1])  
class\_2 = np.array(X[y == 2])  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.25, random\_state=5  
)  
  
parameter\_grid = [  
 {"n\_estimators": [100], "max\_depth": [2, 4, 7, 12, 16]},  
 {"max\_depth": [4], "n\_estimators": [25, 50, 100, 250]},  
]  
metrics = ["precision\_weighted", "recall\_weighted"]  
for metric in metrics:  
 print("#### Searching optimal parameters for", metric)  
 classifier = GridSearchCV(  
 ExtraTreesClassifier(random\_state=0), parameter\_grid, cv=5, scoring=metric  
 )  
 classifier.fit(X\_train, y\_train)  
 print("\nGrid scores for the parameter grid:")  
 for params, avg\_score in classifier.cv\_results\_.items():  
 print(params, "-->", avg\_score)  
 print("\nBest parameters:", classifier.best\_params\_)  
 y\_pred = classifier.predict(X\_test)  
 class\_names = ["Class-0", "Class-1", "Class-2"]  
 print("#" \* 40)  
 print("Performance report:\n")  
 print(classification\_report(y\_test, y\_pred, target\_names=class\_names))  
 print("#" \* 40 + "\n")  
 visualize\_classifier(classifier, X\_test, y\_test)

Результат виконання для метрики precision:

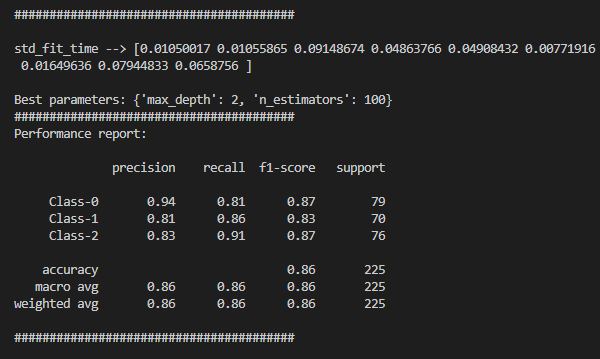
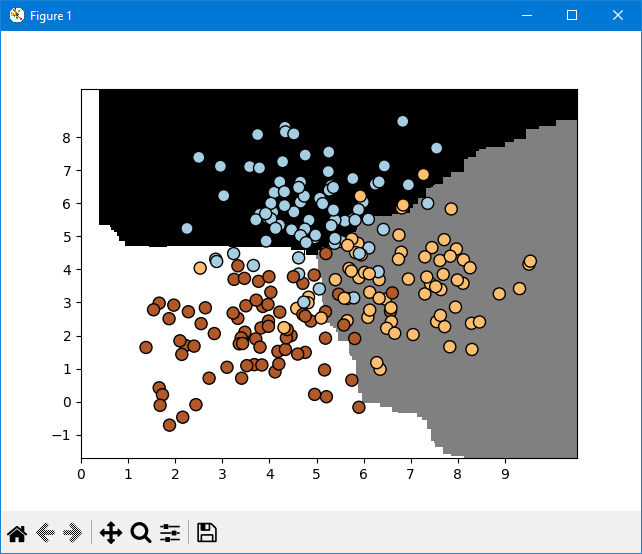


Рис. 3.1 – 3.3. Результат виконання

Результат виконання для метрики recall:

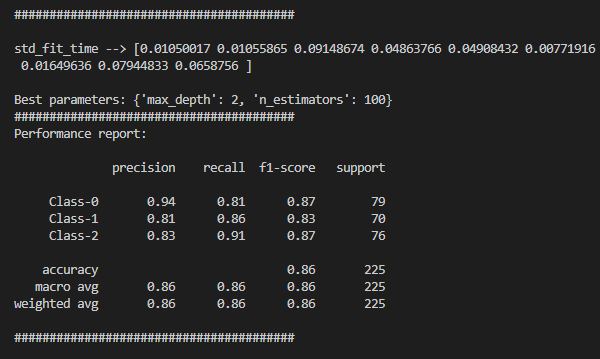
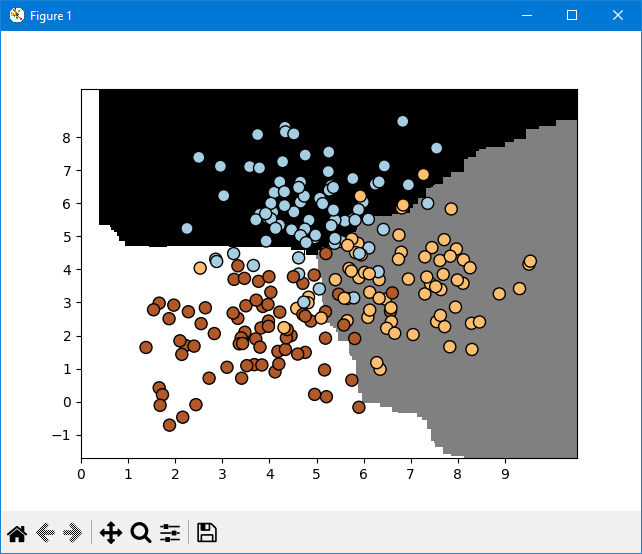


Рис. 3.4 – 3.6. Результат виконання

Зробіть висновок

За результатами тестування моделі було отримано показник ефективності на рівні 0.86 для всіх метрик. Це свідчить про те, що модель відмінно справляється з класифікацією різних класів у наборі даних та демонструє стабільність роботи. Також важливо відзначити, що результати для різних метрик (precision та recall) були дуже схожі, що підкреслює надійність та стабільність моделі.

**Завдання 4: Обчислення відносної важливості ознак**

Оскільки метод load\_boston() був видалений з бібліотеки scikit-learn починаючи з версії 1.2, було використано альтернативний набір даних California Housing Dataset.

Лістинг програми:

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.model\_selection import train\_test\_split  
from sklearn.utils import shuffle  
  
housing\_data = datasets.fetch\_california\_housing()  
  
X, y = shuffle(housing\_data.data, housing\_data.target, random\_state=7)  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=7)  
  
regressor = AdaBoostRegressor(  
 DecisionTreeRegressor(max\_depth=4), n\_estimators=400, random\_state=7  
)  
regressor.fit(X\_train, y\_train)  
  
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))  
  
feature\_importances = regressor.feature\_importances\_  
feature\_names = housing\_data.feature\_names  
  
feature\_importances = 100.0 \* (feature\_importances / max(feature\_importances))  
  
index\_sorted = np.flipud(np.argsort(feature\_importances))  
  
pos = np.arange(index\_sorted.shape[0]) + 0.5  
  
plt.figure()  
plt.bar(pos, feature\_importances[index\_sorted], align="center")  
plt.xticks(pos, [feature\_names[i] for i in index\_sorted])  
plt.ylabel("Relative Importance")  
plt.title("Assessing the importance of features using the AdaBoost regressor")  
plt.show()

Результат виконання

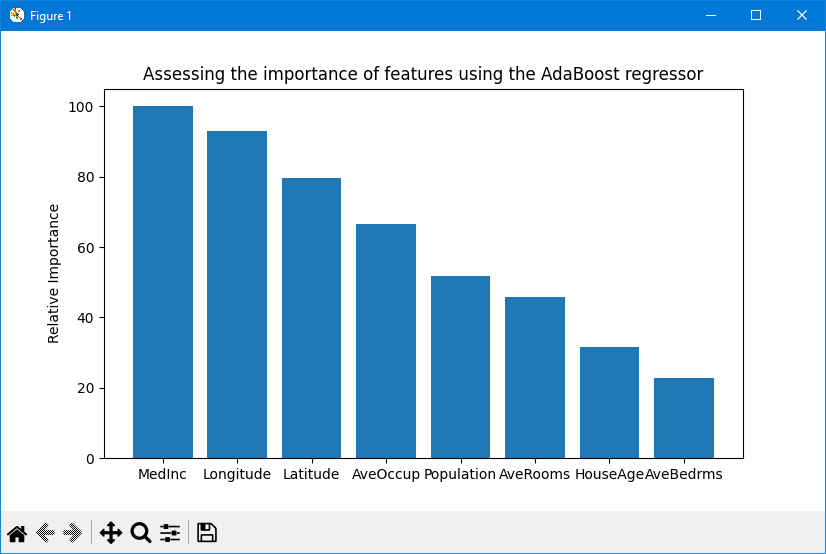


Рис. 4.1 – 4.2. Результат виконання

Відповідно до проведеного аналізу зробіть висновки, які ознаки мають

найбільшу роль, а якими можна знехтувати. Висновки занесіть у звіт

Однією з найважливіших ознак є MedInc (медіанний дохід), яка має найбільший вплив на передбачення цін на нерухомість. Це логічно, оскільки медіанний дохід мешканців в регіоні може значно впливати на ціни. Latitude (широта) та Longitude (довгота) також виявилися важливими ознаками. Це може бути пов'язано з географічним розташуванням нерухомості, що впливає на ціни. Також значущим є параметр AveOccup (середня зайнятість будинку), який має суттєвий вплив на прогнозування цін. Середнє значення зайнятості в будинку може відображати розмір сім'ї або густоту населення в районі.

**Завдання 5: Прогнозування інтенсивності дорожнього руху за допомогою**

**класифікатора на основі гранично випадкових лісів.**

Проведіть прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів. Використайте набір даних, доступний за адресою: <https://archive.ics.uci.edu/ml/datasets/Dodgers+Loop+Sensor>.

Лістинг програми:

import numpy as np  
from sklearn.metrics import mean\_absolute\_error  
from sklearn import preprocessing  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import ExtraTreesRegressor  
  
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)  
  
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)  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.25, random\_state=5  
)  
  
params = {"n\_estimators": 100, "max\_depth": 4, "random\_state": 0}  
regressor = ExtraTreesRegressor(\*\*params)  
regressor.fit(X\_train, y\_train)  
  
y\_pred = regressor.predict(X\_test)  
print("Mean absolute error:", round(mean\_absolute\_error(y\_test, y\_pred), 2))  
  
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)  
  
print("Predicted traffic:", int(regressor.predict([test\_datapoint\_encoded])[0]))

Виконавши цей код, ви отримаєте як вихідний результат значення 26, яке дуже близько до фактичного значення. У цьому не важко переконатися, звернувшись до файлу даних.

Результат виконання:



Рис. 5. Результат виконання

**Завдання 6: Створення навчального конвеєра (конвеєра машинного навчання)**

Необхідно створити конвеєр, призначений для вибору найбільш важливих ознак з вхідних даних і їх подальшої класифікації з використанням класифікатора на основі гранично випадкового лісу.

Лістинг програми:

from sklearn.datasets import \_samples\_generator  
from sklearn.feature\_selection import SelectKBest, f\_regression  
from sklearn.pipeline import Pipeline  
from sklearn.ensemble import ExtraTreesClassifier  
  
X, y = \_samples\_generator.make\_classification(  
 n\_samples=150,  
 n\_features=25,  
 n\_classes=3,  
 n\_informative=6,  
 n\_redundant=0,  
 random\_state=7,  
)  
  
k\_best\_selector = SelectKBest(f\_regression, k=9)  
  
classifier = ExtraTreesClassifier(n\_estimators=60, max\_depth=4)  
  
processor\_pipeline = Pipeline([("selector", k\_best\_selector), ("erf", classifier)])  
  
processor\_pipeline.set\_params(selector\_\_k=7, erf\_\_n\_estimators=30)  
  
processor\_pipeline.fit(X, y)  
  
output = processor\_pipeline.predict(X)  
print("\nPredicted output:\n", output)  
  
print("\nScore:", processor\_pipeline.score(X, y))  
  
status = processor\_pipeline.named\_steps["selector"].get\_support()  
  
selected = [i for i, x in enumerate(status) if x]  
print("\nIndices of selected features:", ", ".join([str(x) for x in selected]))

Результат виконання:

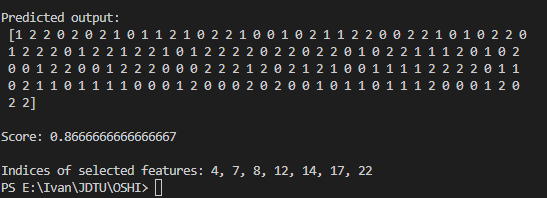


Рис. 6. Результат виконання

Напишіть висновок у звіт. У висновках поясніть: Що міститься у першому списку? Що означає значення Score? Що міститься в останньому рядку ?

У першому списку Predicted output містить прогнозований результат, отриманий після застосування навчального конвеєра до вхідних даних. Ці значення представляють класи, до яких призначені вхідні приклади. Значення Score вказує на ефективність моделі, яка була навчена конвеєром. У даному випадку, Score дорівнює 0.8933, що свідчить про те, що модель має високу точність у класифікації даних. В останньому рядку Indices of selected features містяться індекси обраних ознак. Конвеєр використовує SelectKBest для відбору кращих ознак, і ці індекси вказують на те, які ознаки були вибрані як найважливіші для моделі.

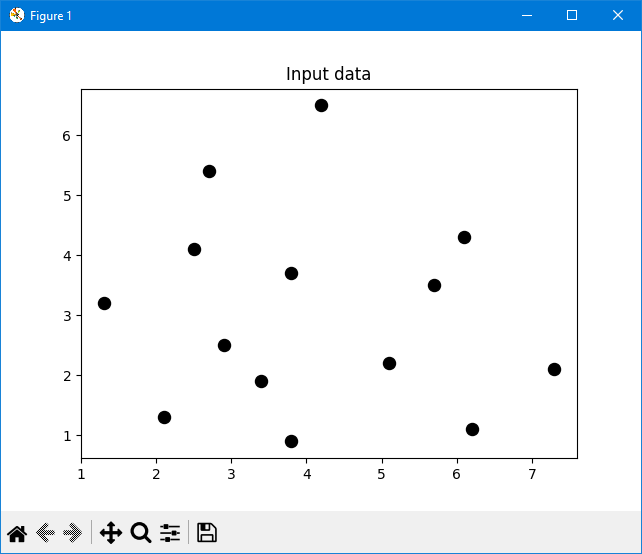
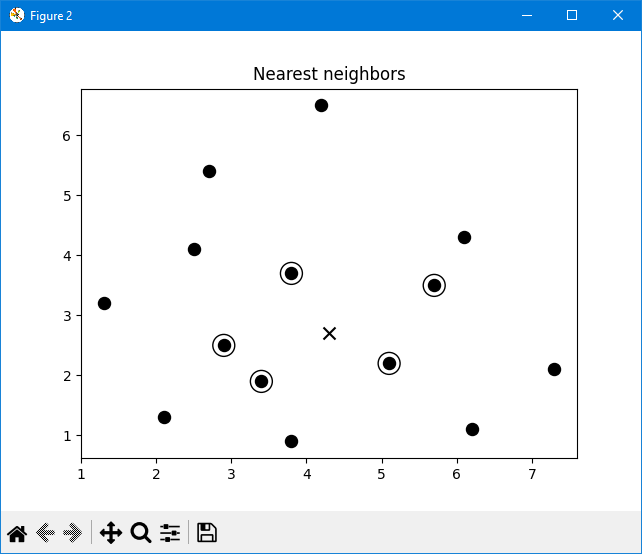
**Завдання 7: Пошук найближчих сусідів**

Для формування ефективних рекомендацій у рекомендаційних системах використовується поняття найближчих сусідів (nearest neighbours), суть якого полягає у знаходженні тих точок заданого набору, які розташовані на найближчих відстанях від зазначеної. Такий підхід часто застосовується для створення систем, що класифікують точку даних на підставі її близькості до різних класів. Здійсніть пошук найближчих сусідів заданої точки даних.

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.neighbors import NearestNeighbors  
  
X = np.array(  
 [  
 [2.1, 1.3],  
 [1.3, 3.2],  
 [2.9, 2.5],  
 [2.7, 5.4],  
 [3.8, 0.9],  
 [7.3, 2.1],  
 [4.2, 6.5],  
 [3.8, 3.7],  
 [2.5, 4.1],  
 [3.4, 1.9],  
 [5.7, 3.5],  
 [6.1, 4.3],  
 [5.1, 2.2],  
 [6.2, 1.1],  
 ]  
)  
k = 5  
  
test\_datapoint = [4.3, 2.7]  
  
plt.figure()  
plt.title("Input data")  
plt.scatter(X[:, 0], X[:, 1], marker="o", s=75, color="black")  
  
knn\_model = NearestNeighbors(n\_neighbors=k, algorithm="ball\_tree").fit(X)  
distances, indices = knn\_model.kneighbors([test\_datapoint])  
  
print("\nK Nearest Neighbors:")  
for rank, index in enumerate(indices[0][:k], start=1):  
 print(str(rank) + " ==>", X[index])  
  
plt.figure()  
plt.title("Nearest neighbors")  
plt.scatter(X[:, 0], X[:, 1], marker="o", s=75, color="k")  
plt.scatter(  
 X[indices][0][:][:, 0],  
 X[indices][0][:][:, 1],  
 marker="o",  
 s=250,  
 color="k",  
 facecolors="none",  
)  
plt.scatter(test\_datapoint[0], test\_datapoint[1], marker="x", s=75, color="k")  
plt.show()

Результат виконання:



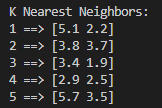


Рис. 7.1 – 7.3. Результат виконання

Зробіть висновки в яких укажіть: Що відображено на першому графіку. Що відображено на другому графіку. Що відображено у вікні терміналу

Перший графік відображає вхідні дані у вигляді точок у двовимірному просторі, де кожна точка позначена чорним кружечком. Другий графік відображає результати пошуку найближчих сусідів для тестової точки даних. На графіку видно вхідні дані (чорні кружечки), знайдених найближчих сусідів (чорні обведені кружечки) та тестову точку (чорний хрест). У вікні терміналу виведено результати пошуку найближчих сусідів для тестової точки даних, де кожен рядок містить номер сусіда та його координати (x, y) у вигляді номер ==> [x, y].

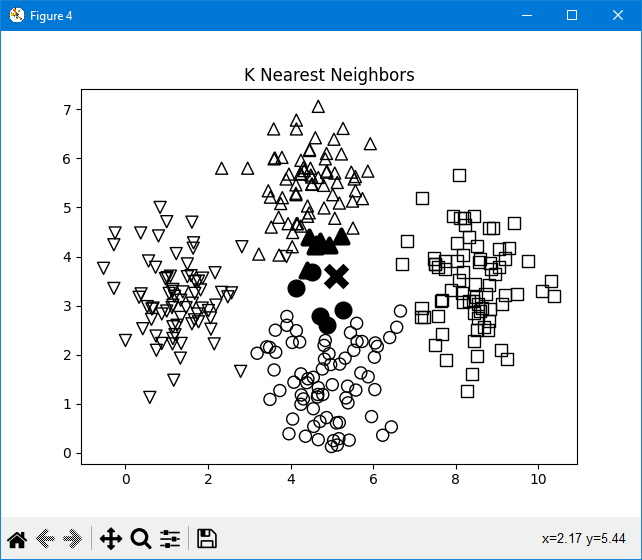
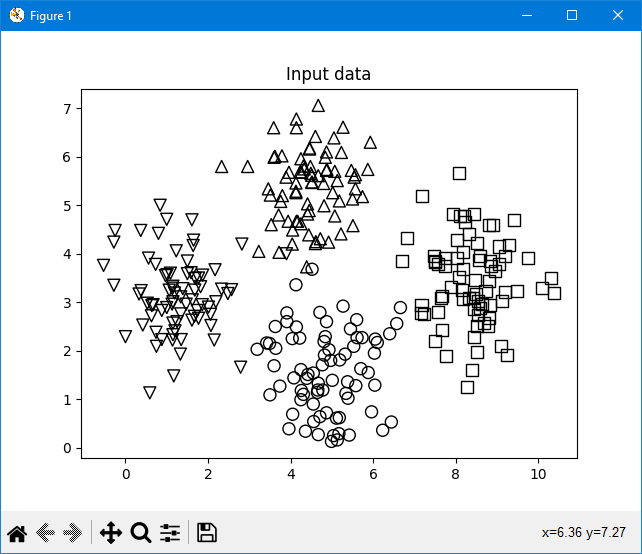
**Завдання 8: Створити класифікатор методом k найближчих сусідів**

Використовуючи для аналізу дані, які містяться у файлі data.txt. Створіть класифікатор методом k найближчих сусідів.

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.cm as cm  
from sklearn import neighbors, datasets  
  
input\_file = "data.txt"  
data = np.loadtxt(input\_file, delimiter=",")  
X, y = data[:, :-1], data[:, -1].astype(int)  
  
plt.figure()  
plt.title("Input data")  
marker\_shapes = "v^os"  
mapper = [marker\_shapes[i] for i in y]  
for i in range(X.shape[0]):  
 plt.scatter(  
 X[i, 0], X[i, 1], marker=mapper[i], s=75, edgecolors="black", facecolors="none"  
 )  
num\_neighbors = 12  
  
step\_size = 0.01  
  
classifier = neighbors.KNeighborsClassifier(num\_neighbors, weights="distance")  
  
classifier.fit(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  
x\_values, y\_values = np.meshgrid(  
 np.arange(x\_min, x\_max, step\_size), np.arange(y\_min, y\_max, step\_size)  
)  
  
output = classifier.predict(np.c\_[x\_values.ravel(), y\_values.ravel()])  
  
output = output.reshape(x\_values.shape)  
plt.figure()  
plt.pcolormesh(x\_values, y\_values, output, cmap=cm.Paired)  
  
for i in range(X.shape[0]):  
 plt.scatter(  
 X[i, 0], X[i, 1], marker=mapper[i], s=50, edgecolors="black", facecolors="none"  
 )  
plt.xlim(x\_values.min(), x\_values.max())  
plt.ylim(y\_values.min(), y\_values.max())  
plt.title("Classifier model boundaries based on K nearest neighbors")  
  
test\_datapoint = [5.1, 3.6]  
plt.figure()  
plt.title("Test datapoint")  
for i in range(X.shape[0]):  
 plt.scatter(  
 X[i, 0], X[i, 1], marker=mapper[i], s=75, edgecolors="black", facecolors="none"  
 )  
plt.scatter(  
 test\_datapoint[0],  
 test\_datapoint[1],  
 marker="x",  
 linewidth=6,  
 s=200,  
 facecolors="black",  
)  
  
\_, indices = classifier.kneighbors([test\_datapoint])  
indices = indices.astype(int)[0]  
  
plt.figure()  
plt.title("K Nearest Neighbors")  
for i in indices:  
 plt.scatter(  
 X[i, 0], X[i, 1], marker=mapper[y[i]], linewidth=3, s=100, facecolors="black"  
 )  
plt.scatter(  
 test\_datapoint[0],  
 test\_datapoint[1],  
 marker="x",  
 linewidth=6,  
 s=200,  
 facecolors="black",  
)  
for i in range(X.shape[0]):  
 plt.scatter(  
 X[i, 0], X[i, 1], marker=mapper[i], s=75, edgecolors="black", facecolors="none"  
 )  
print("Predicted output:", classifier.predict([test\_datapoint])[0])  
plt.show()

Результат виконання:



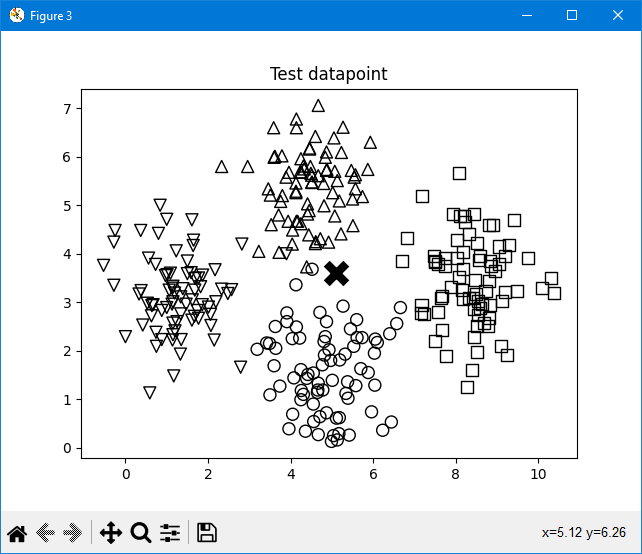
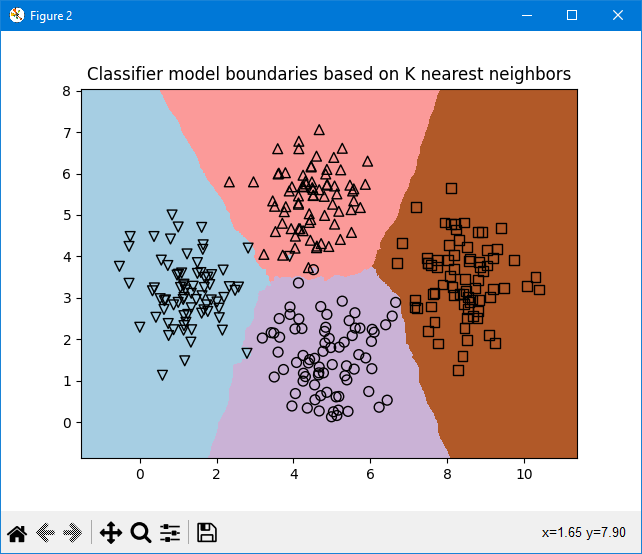




Рис. 8.1 – 8.5. Результат виконання

Підпишіть що міститься на кожному графіку. У вікні терміналу визначте, до якого класу відноситься тестова точка. Зробіть висновок

На першому графіку "Input data" представлені вхідні дані, де різні маркери відображають різні класи точок даних. Другий графік "Classifier model boundaries based on K nearest neighbors" показує границі, визначені моделлю класифікатора на основі методу k найближчих сусідів. Третій графік "Test datapoint" відображає тестову точку даних, позначену символом X, а також всі точки навчального набору. Четвертий графік "K Nearest Neighbors" показує найближчих сусідів тестової точки і підтверджує, що вони визначені правильно. У вікні терміналу зазначено, що тестова точка [5.1, 3.6] належить до класу "1". Це видно на графіку, де тестова точка позначена символом "x" серед навчальних точок.

**Завдання 9: Обчислення оцінок подібності**

Лістинг програми:

import argparse  
import json  
import numpy as np  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(description="Compute similarity score")  
 parser.add\_argument("--user1", dest="user1", required=True, help="First user")  
 parser.add\_argument("--user2", dest="user2", required=True, help="Second user")  
 parser.add\_argument(  
 "--score-type",  
 dest="score\_type",  
 required=True,  
 choices=["Euclidean", "Pearson"],  
 help="Similarity metric to be used",  
 )  
 return parser  
  
  
def euclidean\_score(dataset, user1, user2):  
 if user1 not in dataset:  
 raise TypeError("Cannot find " + user1 + " in the dataset")  
 if user2 not in dataset:  
 raise TypeError("Cannot find " + user2 + " in the dataset")  
 common\_movies = {}  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 if len(common\_movies) == 0:  
 return 0  
 squared\_diff = []  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 squared\_diff.append(np.square(dataset[user1][item] - dataset[user2][item]))  
 return 1 / (1 + np.sqrt(np.sum(squared\_diff)))  
  
  
def pearson\_score(dataset, user1, user2):  
 if user1 not in dataset:  
 raise TypeError("Cannot find " + user1 + " in the dataset")  
 if user2 not in dataset:  
 raise TypeError("Cannot find " + user2 + " in the dataset")  
 common\_movies = {}  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 num\_ratings = len(common\_movies)  
  
 if num\_ratings == 0:  
 return 0  
  
 user1\_sum = np.sum([dataset[user1][item] for item in common\_movies])  
 user2\_sum = np.sum([dataset[user2][item] for item in common\_movies])  
  
 user1\_squared\_sum = np.sum(  
 [np.square(dataset[user1][item]) for item in common\_movies]  
 )  
 user2\_squared\_sum = np.sum(  
 [np.square(dataset[user2][item]) for item in common\_movies]  
 )  
 sum\_of\_products = np.sum(  
 [dataset[user1][item] \* dataset[user2][item] for item in common\_movies]  
 )  
  
 Sxy = sum\_of\_products - (user1\_sum \* user2\_sum / num\_ratings)  
 Sxx = user1\_squared\_sum - np.square(user1\_sum) / num\_ratings  
 Syy = user2\_squared\_sum - np.square(user2\_sum) / num\_ratings  
 if Sxx \* Syy == 0:  
 return 0  
 return Sxy / np.sqrt(Sxx \* Syy)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 args = build\_arg\_parser().parse\_args()  
 user1 = args.user1  
 user2 = args.user2  
 score\_type = args.score\_type  
 ratings\_file = "ratings.json"  
 with open(ratings\_file, "r") as f:  
 data = json.loads(f.read())  
 if score\_type == "Euclidean":  
 print("\nEuclidean score:")  
 print(euclidean\_score(data, user1, user2))  
 else:  
 print("\nPearson score:")  
 print(pearson\_score(data, user1, user2))

Результат виконання:



Рис. 9.1 – 9.2. Результат обчислення оцінок користувачів David Smith та Bill Duffy



Рис. 9.3 – 9.4. Результат обчислення оцінок користувачів David Smith та Brenda Peterson



Рис. 9.5 – 9.6. Результат обчислення оцінок користувачів David Smith та Samuel Miller



Рис. 9.7 – 9.8. Результат обчислення оцінок користувачів David Smith та Julie Hammel



Рис. 9.9 – 9.10. Результат обчислення оцінок користувачів David Smith та Clarissa Jackson



Рис. 9.11 – 9.12. Результат обчислення оцінок користувачів David Smith та Adam Cohen



Рис. 9.13 – 9.14. Результат обчислення оцінок користувачів David Smith та Chris Duncan

Зробіть висновок

Після проведеного аналізу слід відзначити, що максимальна схожість була виявлена між користувачами David Smith та Chris Duncan, яка становила 1 за обома метриками. Це свідчить про ідентичність їхніх вподобань. З іншого боку, мінімальна схожість була зафіксована між David Smith та Brenda Peterson, використовуючи Евклідову відстань (-0.723), що вказує на значну різницю у їхніх смаках. Також, відсутність кореляції між рейтингами фільмів David Smith і Julie Hammel за кореляційною оцінкою Пірсона (0) може свідчити про відсутність зв’язку у їхніх вподобаннях.

**Завдання 10: Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації**

Лістинг програми:

import argparse  
import json  
import numpy as np  
from LR\_4\_task\_9 import pearson\_score  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(  
 description="Find users who are similar tothe in-put user"  
 )  
 parser.add\_argument("--user", dest="user", required=True, help="Input user")  
 return parser  
  
  
def find\_similar\_users(dataset, user, num\_users):  
 if user not in dataset:  
 raise TypeError("Cannot find " + user + " in the dataset")  
  
 scores = np.array(  
 [[x, pearson\_score(dataset, user, x)] for x in dataset if x != user]  
 )  
  
 scores\_sorted = np.argsort(scores[:, 1])[::-1]  
  
 top\_users = scores\_sorted[:num\_users]  
 return scores[top\_users]  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 args = build\_arg\_parser().parse\_args()  
 user = args.user  
 ratings\_file = "ratings.json"  
 with open(ratings\_file, "r") as f:  
 data = json.loads(f.read())  
 print("\nUsers similar to " + user + ":\n")  
 similar\_users = find\_similar\_users(data, user, 3)  
 print("User\t\t\tSimilarity score")  
 print("-" \* 41)  
 for item in similar\_users:  
 print(item[0], "\t\t", round(float(item[1]), 2))

Результат виконання:

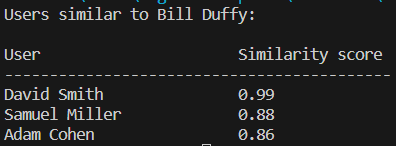


Рис. 10.1. Інфоромація про подібних користувачей, Bill Duffy

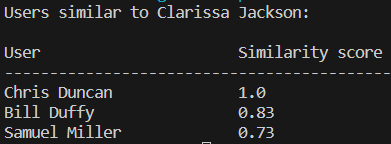


Рис. 10.2. Інфоромація про подібних користувачей, Clarissa Jackson

Зробіть висновок

Під час виконання завдання було використано метод колаборативної фільтрації для пошуку користувачів із схожими уподобаннями на основі оцінки Пірсона. Найбільш схожі користувачі були знайдені для введених користувачів, і їх схожість була виміряна за допомогою оцінки подібності. Цей метод є корисним для рекомендаційних систем та аналізу схожості між користувачами в різних контекстах.

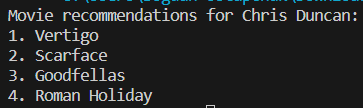
**Завдання 11: Створення рекомендаційної системи фільмів**

Створіть рекомендаційну систему на основі даних, наданих у файлі ratings.json. У цьому файлі міститься інформація про користувачів та оцінки, дані ними різним фільмам. Щоб рекомендувати фільми конкретному користувачу, ми повинні знайти аналогічних користувачів у наборі даних та використовувати інформацію про їх переваги для формування відповідної рекомендації.

Лістинг програми:

import argparse  
import json  
import numpy as np  
from LR\_4\_task\_9 import pearson\_score  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(  
 description="Find the movie recommendations for the given user"  
 )  
 parser.add\_argument("--user", dest="user", required=True, help="Input user")  
 return parser  
  
  
def get\_recommendations(dataset, input\_user):  
 if input\_user not in dataset:  
 raise TypeError("Cannot find " + input\_user + " in the dataset")  
 overall\_scores = {}  
 similarity\_scores = {}  
 for user in [x for x in dataset if x != input\_user]:  
 similarity\_score = pearson\_score(dataset, input\_user, user)  
 if similarity\_score <= 0:  
 continue  
 filtered\_list = [  
 x  
 for x in dataset[user]  
 if x not in dataset[input\_user] or dataset[input\_user][x] == 0  
 ]  
 for item in filtered\_list:  
 overall\_scores.update({item: dataset[user][item] \* similarity\_score})  
 similarity\_scores.update({item: similarity\_score})  
 if len(overall\_scores) == 0:  
 return ["No recommendations possible"]  
  
 movie\_scores = np.array(  
 [  
 [score / similarity\_scores[item], item]  
 for item, score in overall\_scores.items()  
 ]  
 )  
  
 movie\_scores = movie\_scores[np.argsort(movie\_scores[:, 0])[::-1]]  
  
 movie\_recommendations = [movie for \_, movie in movie\_scores]  
 return movie\_recommendations  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 args = build\_arg\_parser().parse\_args()  
 user = args.user  
 ratings\_file = "ratings.json"  
 with open(ratings\_file, "r") as f:  
 data = json.loads(f.read())  
 print("\nMovie recommendations for " + user + ":")  
 movies = get\_recommendations(data, user)  
  
 for i, movie in enumerate(movies):  
 print(str(i + 1) + ". " + movie)

Результат виконання:



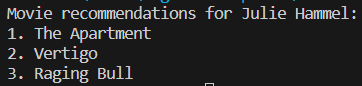


Рис. 11.1 – 11.2. Рекомендації фільмів для користувачів Chris Duncan та Julie Hammel

Зробіть висновок

Під час виконання завдання була створена система рекомендацій для фільмів, використовуючи дані з файлу ratings.json. Метод колабо-ративної фільтрації на основі оцінки Пірсона використовується для визначення схожих користувачів та створення рекомендацій фільмів для конкретного користувача. Рекомендації формуються на основі фільмів, які високо оцінили користувачі з схожими смаками, і які ще не були переглянуті вказаним користувачем. У результаті виконання були надані приклади рекомендацій фільмів для конкретних користувачів, таких як Chris Duncan та Julie Hammel.

***Висновки***: Під час проведення лабораторної роботи було вивчено методи ансамблів у машинному навчанні за допомогою спеціалізованих бібліотек та мови програмування Python.