ЛАБОРАТОРНА РОБОТА №4

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

Мета: використовуючи спеціалізовані бібліотеки та мову програмування Python, дослідити методи ансамблів у машинному навчанні та створити рекомендаційні системи.

Посилання на проект: https://github.com/ipz202-rev/AI-lab4

Хід роботи:

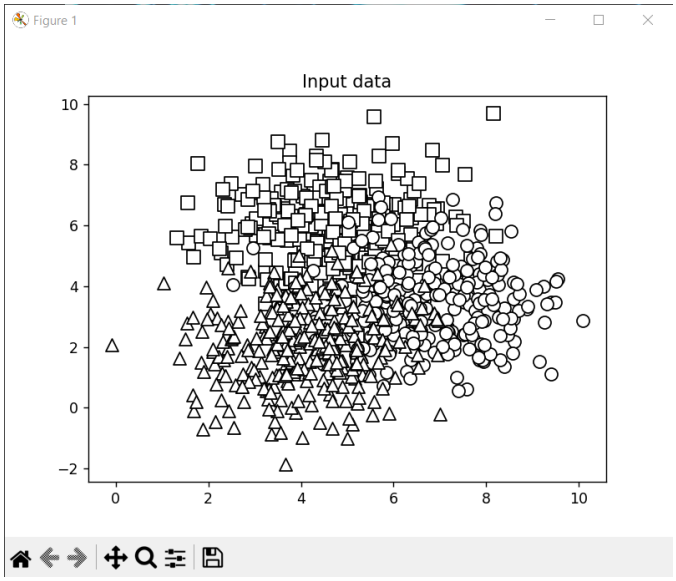
Завдання №4.1. Створення класифікаторів на основі випадкових та гранично випадкових лісів.

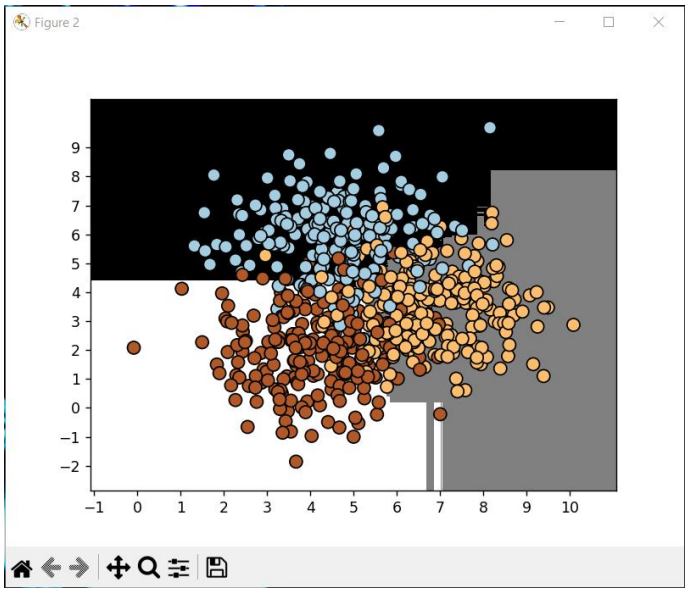
Використовувати файл вхідних даних: data_random_forests.txt, побудувати класифікатори на основі випадкових та гранично випадкових лісів.

Лістинг файлу LR_4_task_1.py:

```
import argparse
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
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


# Парсер аргументів
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__':
    # Вилучення вхідних аргументів
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type
```

```
    # Завантаження вхідних даних
    input_file = 'data_random_forests.txt'
    data = np.loadtxt(input_file, delimiter=',')
    X, y = data[:, :-1], data[:, -1]

    # Розбиття вхідних даних на три класи
    class_0 = np.array(X[y == 0])
    class_1 = np.array(X[y == 1])
    class_2 = np.array(X[y == 2])

    # Візуалізація вхідних даних
    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')

    # Розбивка 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 ансамблевого навчання
    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)

    # Перевірка роботи класифікатора
    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")

    # Обчислення параметрів довірливості
    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)

    # Візуалізація точок даних
    visualize_classifier(classifier, test_datapoints, [0] * len(test_datapoints))
    plt.show()
```





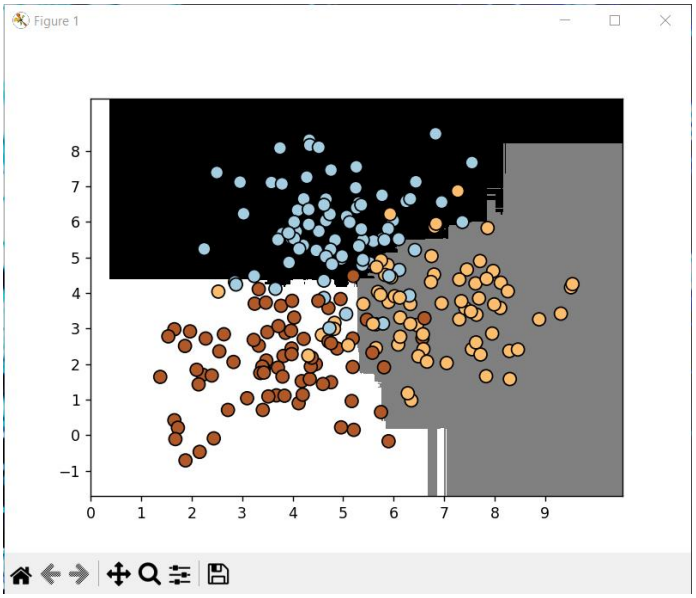
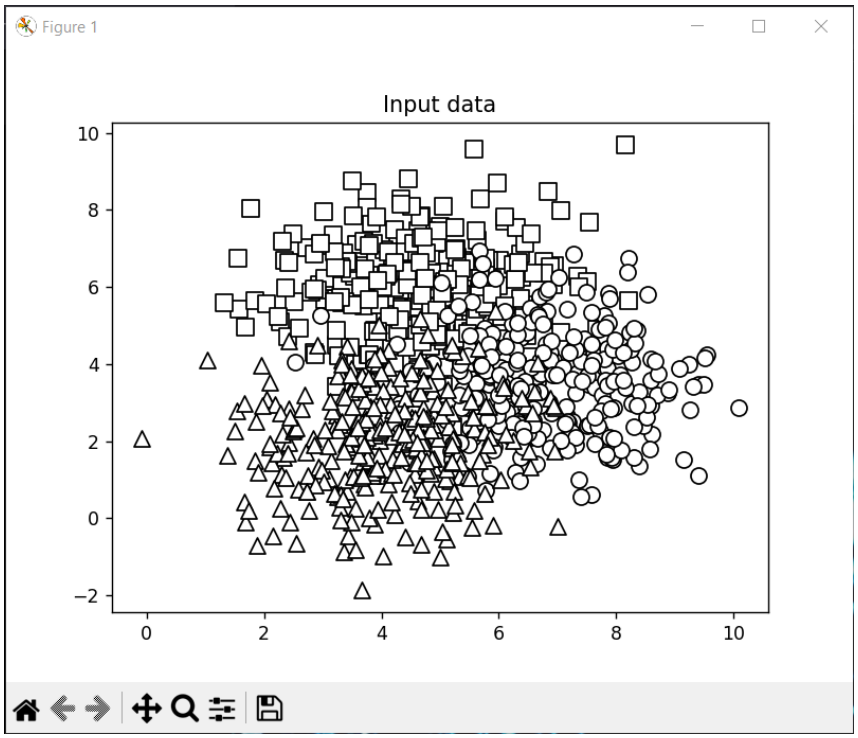
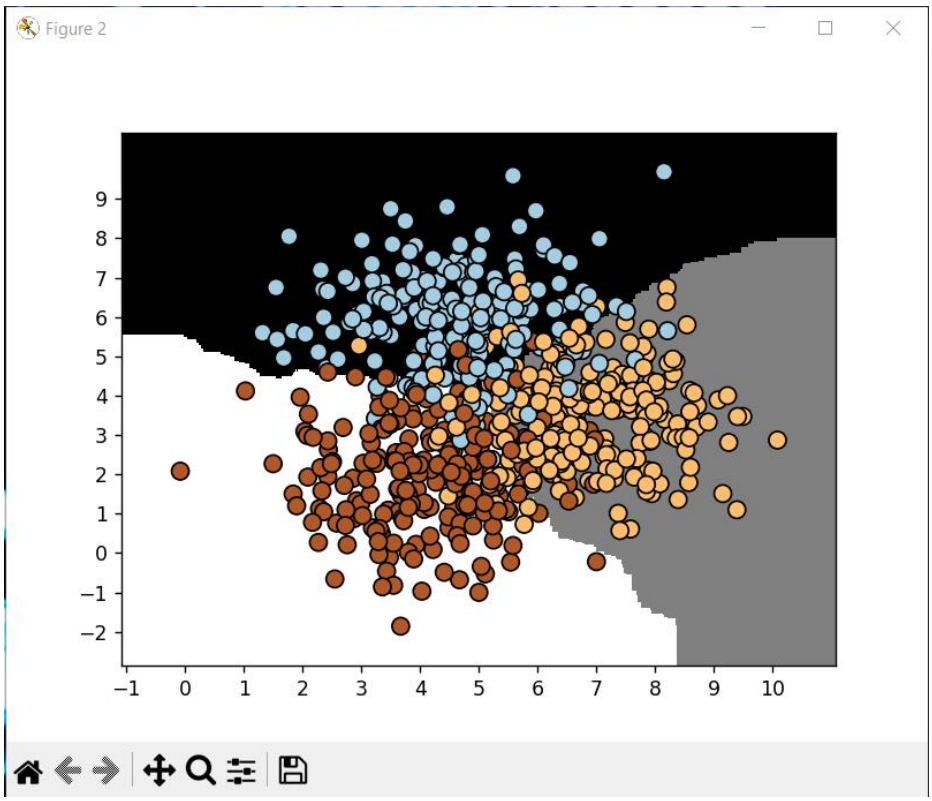


Рис.4.1.1 – 4.1.3. Результат виконання завдання, використовуючи класифікатор на основі випадкового лісу.

```
(venv) PS D:\course-4\semester-1\ai\lab4_project> python LR_4_task_1.py --classifier-type rf

########################################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.91      0.86      0.88       221
     Class-1       0.84      0.87      0.86       230
     Class-2       0.86      0.87      0.86       224

    accuracy                           0.87       675
   macro avg       0.87      0.87      0.87       675
weighted avg       0.87      0.87      0.87       675

########################################

########################################

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.92      0.85      0.88        79
     Class-1       0.86      0.84      0.85        70
     Class-2       0.84      0.92      0.88        76

    accuracy                           0.87       225
   macro avg       0.87      0.87      0.87       225
weighted avg       0.87      0.87      0.87       225

########################################

(venv) PS D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.1.4. Характеристики класифікатора на основі випадкового лісу.





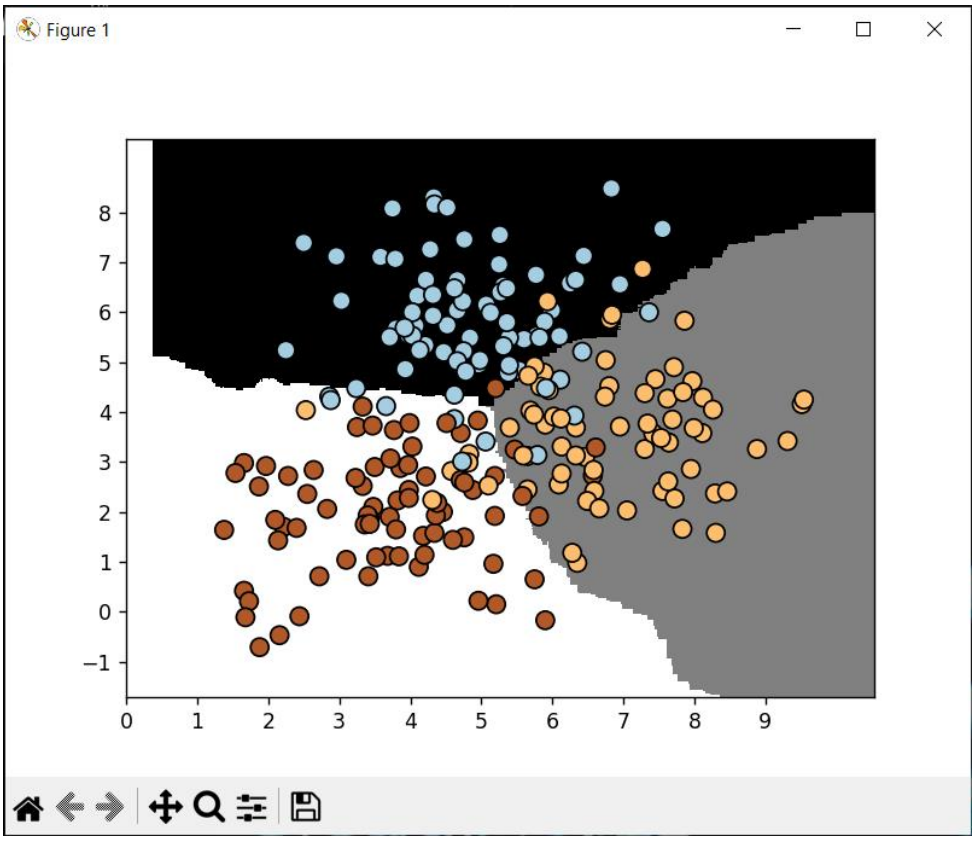


Рис.4.1.5 – 4.1.7. Результат виконання завдання, використовуючи класифікатор на основі гранично випадкового лісу.

```
(venv) PS D:\course-4\semester-1\ai\lab4_project> python LR_4_task_1.py --classifier-type erf

#####################################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.89      0.83      0.86       221
     Class-1       0.82      0.84      0.83       230
     Class-2       0.83      0.86      0.85       224

    accuracy                           0.85       675
   macro avg       0.85      0.85      0.85       675
weighted avg       0.85      0.85      0.85       675

#####################################

#####################################

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.92      0.85      0.88        79
     Class-1       0.84      0.84      0.84        70
     Class-2       0.85      0.92      0.89        76

    accuracy                           0.87       225
   macro avg       0.87      0.87      0.87       225
weighted avg       0.87      0.87      0.87       225

#####################################

(venv) PS D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.1.8. Характеристики класифікатора на основі гранично випадкового лісу.

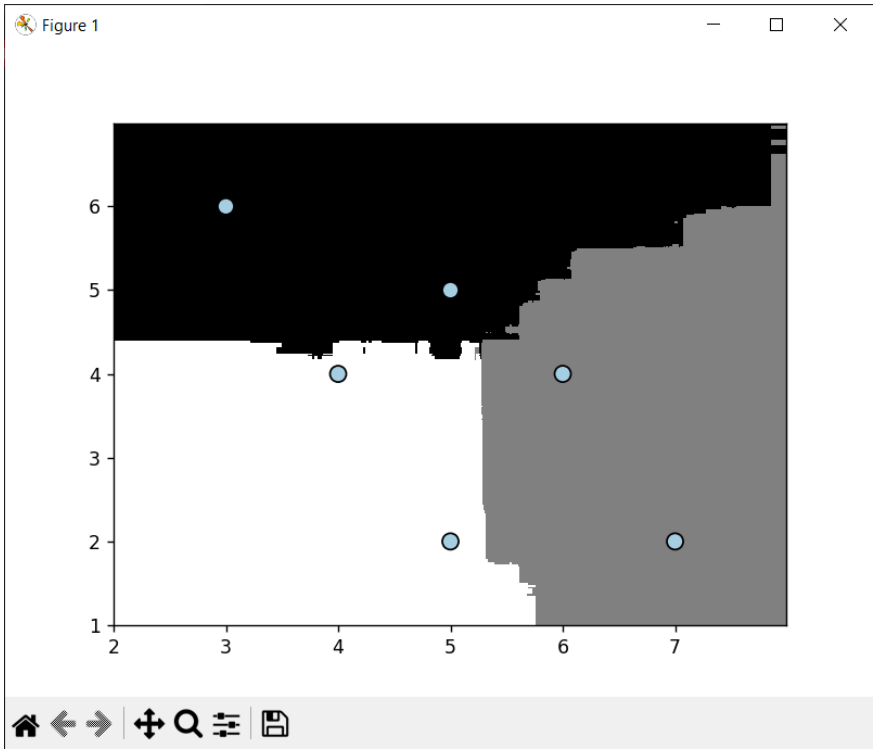


Рис.4.1.9. Візуалізація розподілу тестових точок (на основі випадкового лісу).

```
Confidence measure:

Datapoint: [5 5]
Predicted class: Class-0

Datapoint: [3 6]
Predicted class: Class-0

Datapoint: [6 4]
Predicted class: Class-1

Datapoint: [7 2]
Predicted class: Class-1

Datapoint: [4 4]
Predicted class: Class-2

Datapoint: [5 2]
Predicted class: Class-2
(venv) PS D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.1.10. Розподіл точок на класи.

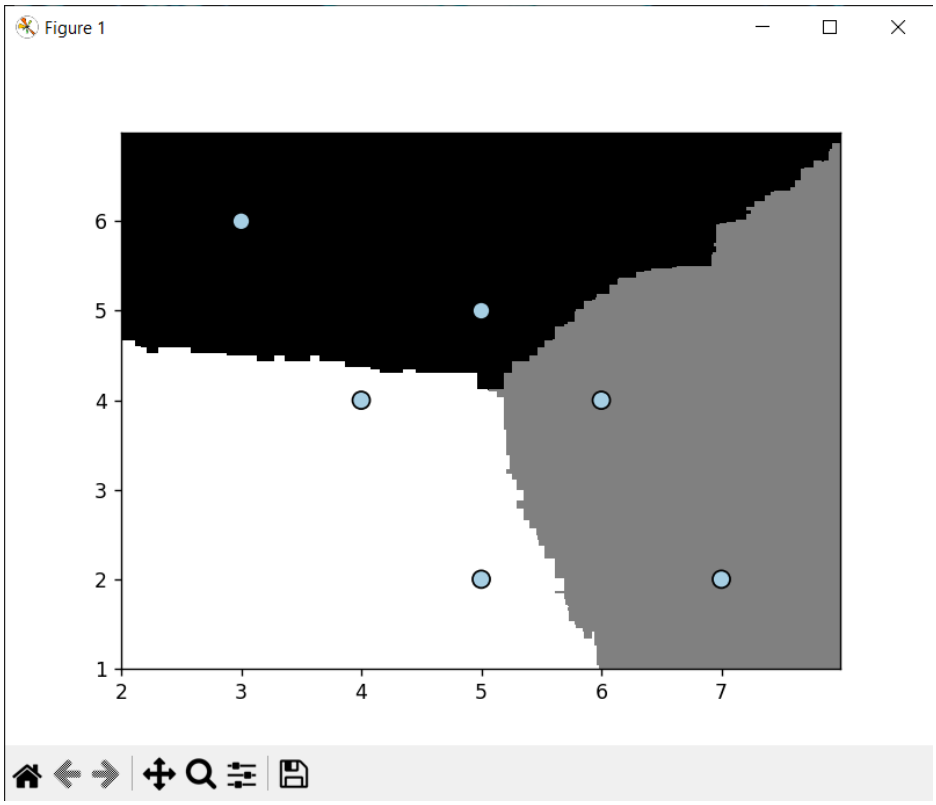


Рис.4.1.11. Візуалізація розподілу тестових точок (на основі граничного випадкового лісу).

```
Confidence measure:

Datapoint: [5 5]
Predicted class: Class-0

Datapoint: [3 6]
Predicted class: Class-0

Datapoint: [6 4]
Predicted class: Class-1

Datapoint: [7 2]
Predicted class: Class-1

Datapoint: [4 4]
Predicted class: Class-2

Datapoint: [5 2]
Predicted class: Class-2
(venv) PS D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.1.12. Розподіл точок на класи.

Враховуючи отримані результати, можна зробити висновок, що у даному завданні обидва методи показали достатньо високі результати. Проте метод граничного випадкового лісу дав кращі границі за рахунок додаткової рандомізації, яка дає більше можливостей для вибору оптимальних дерев рішень.

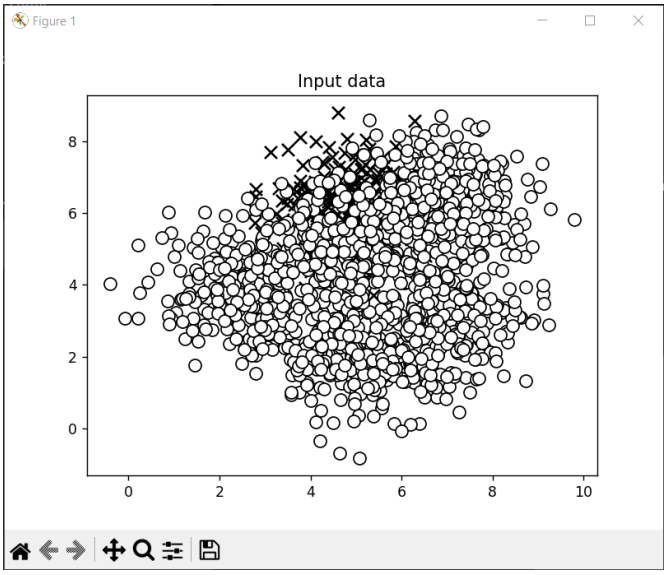
Завдання №4.2. Обробка дисбалансу класів.

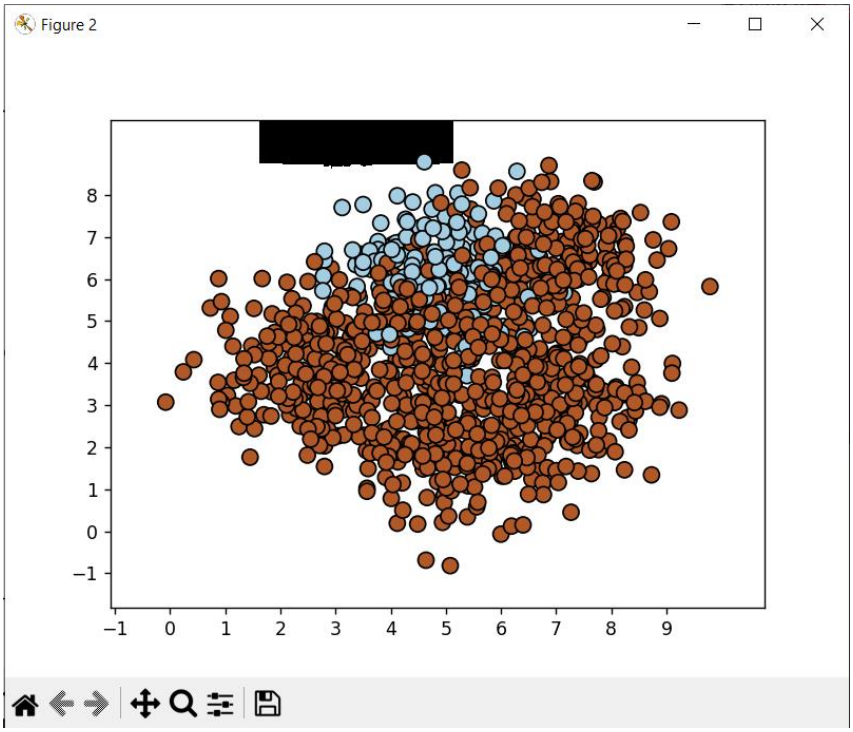
Використовуючи для аналізу дані, які містяться у файлі data_imbalance.txt проведіть обробку з урахуванням дисбалансу класів.

Лістинг файлу LR_4_task_2.py:

```
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

# 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, 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')

# 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)

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, 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()
```





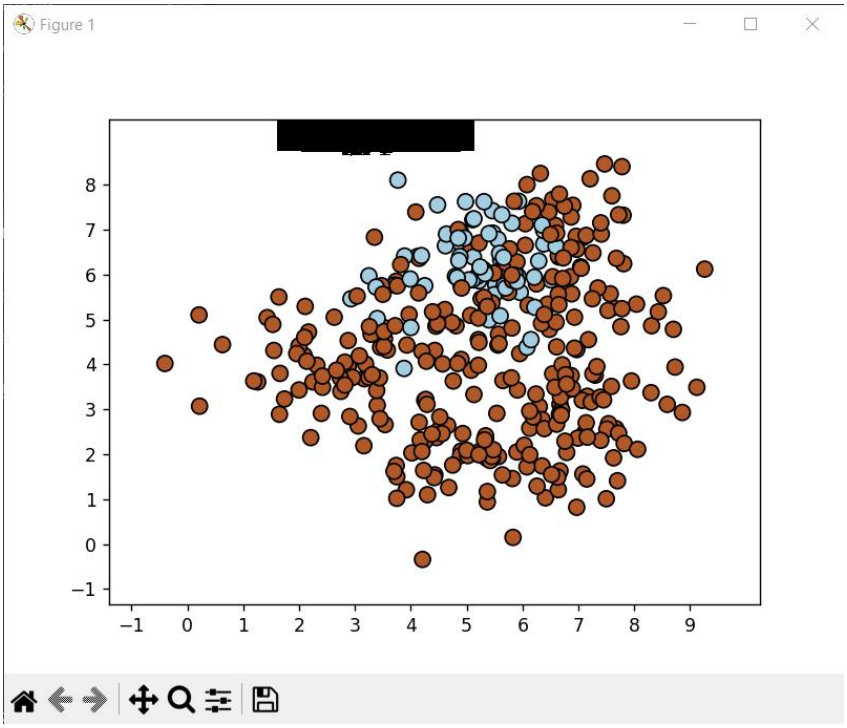
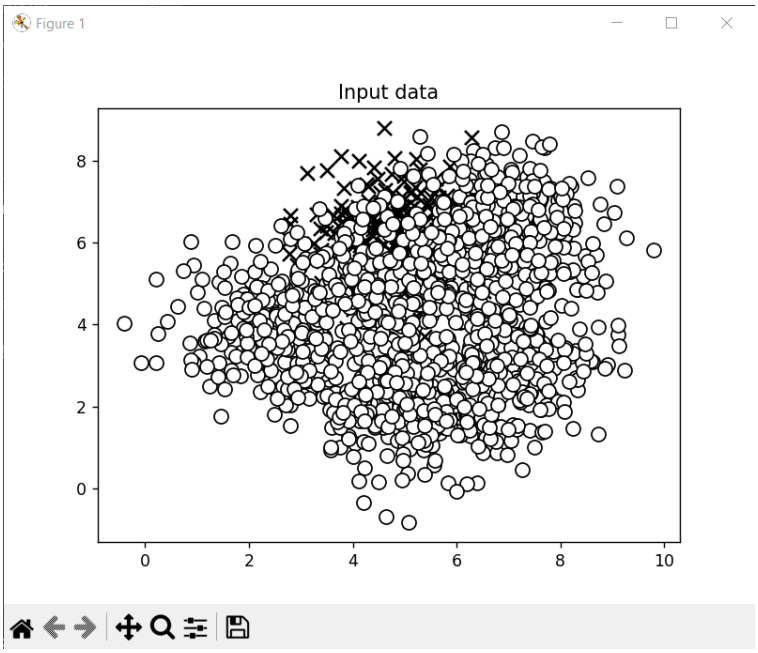
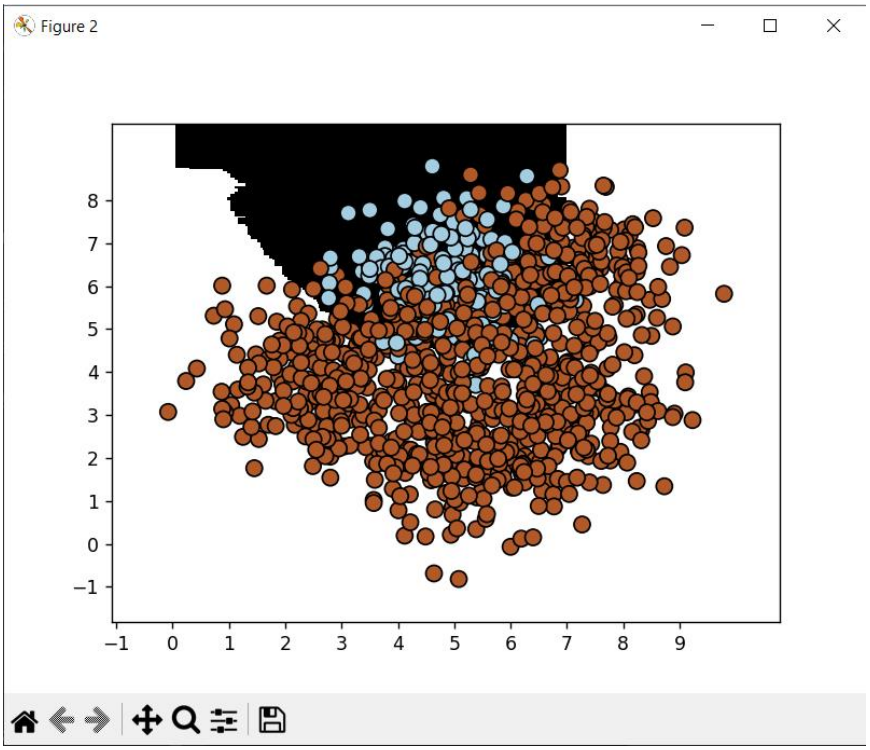


Рис.4.2.1 – 4.2.3. Розподіл незбалансованих даних.

```
(venv) PS D:\course-4\semester-1\ai\lab4_project> python LR_4_task_2.py

########################################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       1.00      0.01      0.01       181
     Class-1       0.84      1.00      0.91       944

    accuracy                           0.84      1125
   macro avg       0.92      0.50      0.46      1125
weighted avg       0.87      0.84      0.77      1125

########################################

########################################

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       1.00      0.00      0.00        69
     Class-1       0.82      1.00      0.90       306

    accuracy                           0.82       375
   macro avg       0.91      0.50      0.45       375
weighted avg       0.85      0.82      0.73       375

########################################

(venv) PS D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.2.4. Характеристики незбалансованої класифікації.





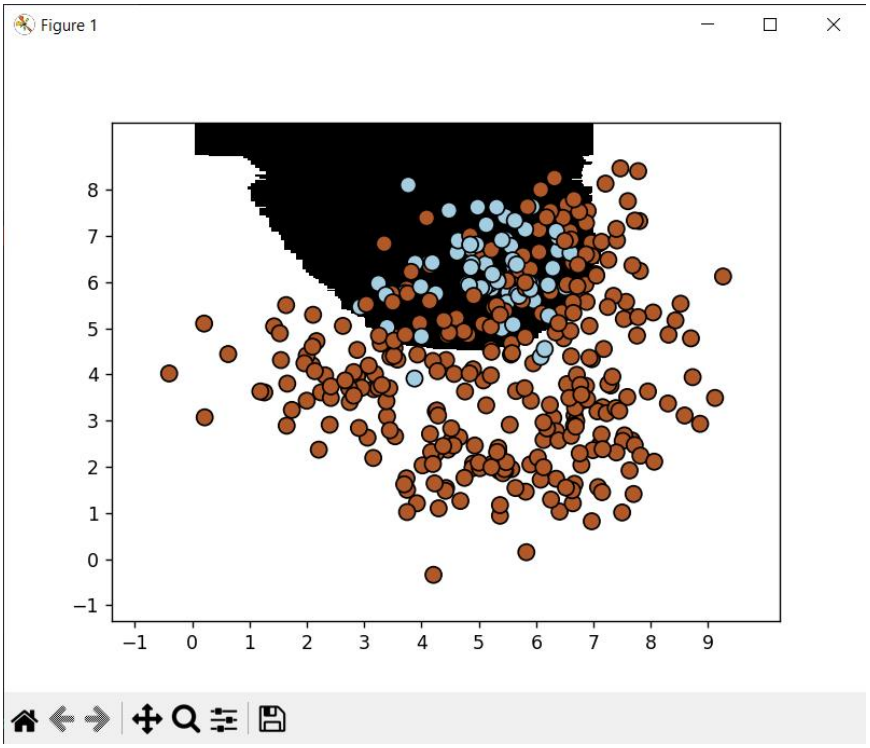


Рис.4.2.5 – 4.2.7. Розподіл збалансованих даних.

```
(venv) PS D:\course-4\semester-1\ai\lab4_project> python LR_4_task_2.py balance

#####################################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.44      0.93      0.60       181
     Class-1       0.98      0.77      0.86       944

    accuracy                           0.80      1125
   macro avg       0.71      0.85      0.73      1125
weighted avg       0.89      0.80      0.82      1125

#####################################

#####################################

Classifier performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.45      0.94      0.61        69
     Class-1       0.98      0.74      0.84       306

    accuracy                           0.78       375
   macro avg       0.72      0.84      0.73       375
weighted avg       0.88      0.78      0.80       375

#####################################

(venv) PS D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.2.8. Характеристики збалансованої класифікації.

При порівнянні отриманих результатів, можна зробити висновок, що після збалансування класів було отримано кращу точність. Це вказує на те, що така модель краще розподіляє дані на класи, враховуючі характеристики усіх з них, а не лише одного.

Завдання №4.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку.

Використовуючи дані, що містяться у файлі data_random_forests.txt, знайти оптимальних навчальних параметрів за допомогою сіткового пошуку.

Лістинг файлу LR_4_task_3.py:

```
import numpy as np
from sklearn.model_selection import cross_val_score, 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]

# 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]}]

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("\nScores across the parameter grid:")
    for params, avg_score in classifier.cv_results_.items():
        print(params, '-->', avg_score)
    print("\nHighest scoring parameter set:", classifier.best_params_)

    y_pred = classifier.predict(X_test)
    class_names = ['Class-0', 'Class-1', 'Class-2']
    print("#" * 40)
    print("Classifier performance on training dataset")
    print(classification_report(y_test, y_pred, target_names=class_names))
    print("#" * 40 + "\n")

    visualize_classifier(classifier, X_test, y_test)
```

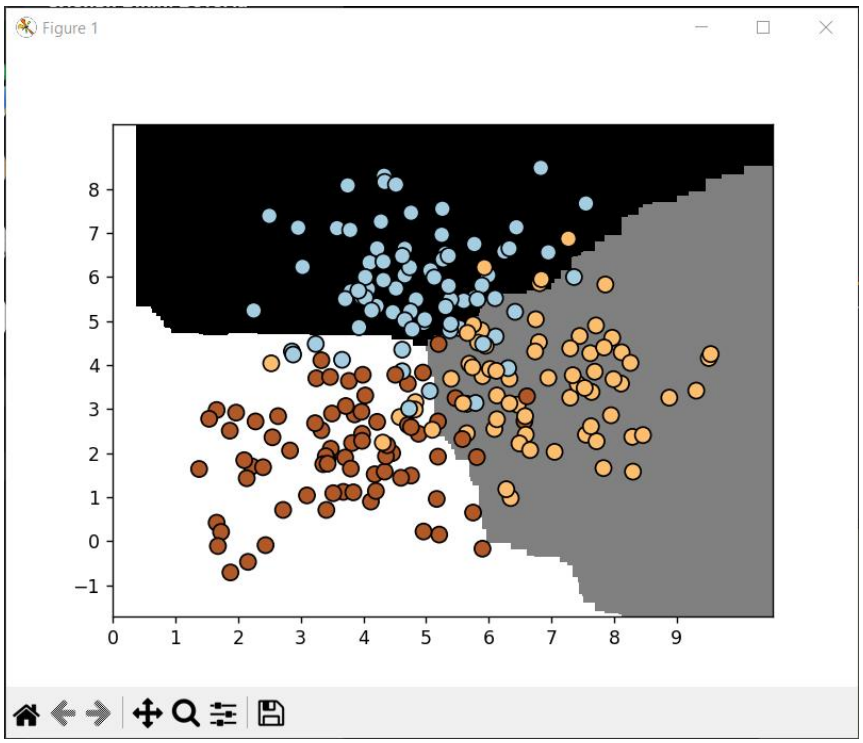


Рис.4.3.1. Візуалізація даних для першої метрики.

```
(venv) PS D:\course-4\semester-1\ai\lab4_project> python LR_4_task_3.py
##### Searching optimal parameters for precision_weighted

Scores for the parameter grid:
mean_fit_time --> [0.09447236 0.07939987 0.09759417 0.11633983 0.12992101 0.01959424
 0.04079518 0.08506846 0.20588112]
std_fit_time --> [0.00779328 0.01078175 0.0174728  0.00592665 0.00784773 0.0031763
 0.00430331 0.00895436 0.01904519]
mean_score_time --> [0.00833001 0.007301   0.01171699 0.01198349 0.00931497 0.0034101
 0.00482378 0.00699434 0.01645403]
std_score_time --> [0.00335544 0.00203352 0.00306151 0.00429093 0.00159099 0.00078108
 0.00117183 0.00171995 0.0053838 ]
param_max_depth --> [2 4 7 12 16 4 4 4 4]
param_n_estimators --> [100 100 100 100 100 25 50 100 250]
params --> [{'max_depth': 2, 'n_estimators': 100}, {'max_depth': 4, 'n_estimators': 100}, {'max_depth': 7,
'n_estimators': 100}, {'max_depth': 12, 'n_estimators': 100}, {'max_depth': 16, 'n_estimators': 100}, {'max
_depth': 4, 'n_estimators': 25}, {'max_depth': 4, 'n_estimators': 50}, {'max_depth': 4, 'n_estimators': 100
}, {'max_depth': 4, 'n_estimators': 250}]
split0_test_score --> [0.87460317 0.85323424 0.85381333 0.81554507 0.80037449 0.87558579
 0.84512618 0.85323424 0.86067019]
split1_test_score --> [0.87694315 0.86737731 0.87662655 0.87651671 0.85190389 0.85594956
 0.85035187 0.86737731 0.87887866]
split2_test_score --> [0.81956872 0.82834119 0.82611079 0.81030267 0.77569734 0.834651
 0.83784535 0.82834119 0.82834119]
split3_test_score --> [0.82078374 0.80260075 0.80192593 0.80351421 0.7942149  0.7931046
 0.80260075 0.80260075 0.80192593]
split4_test_score --> [0.8568842  0.85412387 0.86062409 0.85389639 0.86023157 0.86857693
 0.86332922 0.85412387 0.85412387]
mean_test_score --> [0.8497566  0.84113547 0.84382014 0.83195501 0.81648444 0.84557357
 0.83985067 0.84113547 0.84478797]
std_test_score --> [0.02513165 0.02303185 0.02656029 0.02833428 0.03342873 0.02969796
 0.02040062 0.02303185 0.0268672 ]
rank_test_score --> [1 5 4 8 9 2 7 5 3]

Best parameters: {'max_depth': 2, 'n_estimators': 100}
```

Рис.4.3.2. Сітковий пошук для першої метрики.

```
########################################
Classifier performance on training dataset
              precision    recall  f1-score   support

     Class-0       0.94      0.81      0.87        79
     Class-1       0.81      0.86      0.83        70
     Class-2       0.83      0.91      0.87        76

    accuracy                           0.86       225
   macro avg       0.86      0.86      0.86       225
weighted avg       0.86      0.86      0.86       225

########################################
```

Рис.4.3.3. Оцінка класифікації для першої метрики.

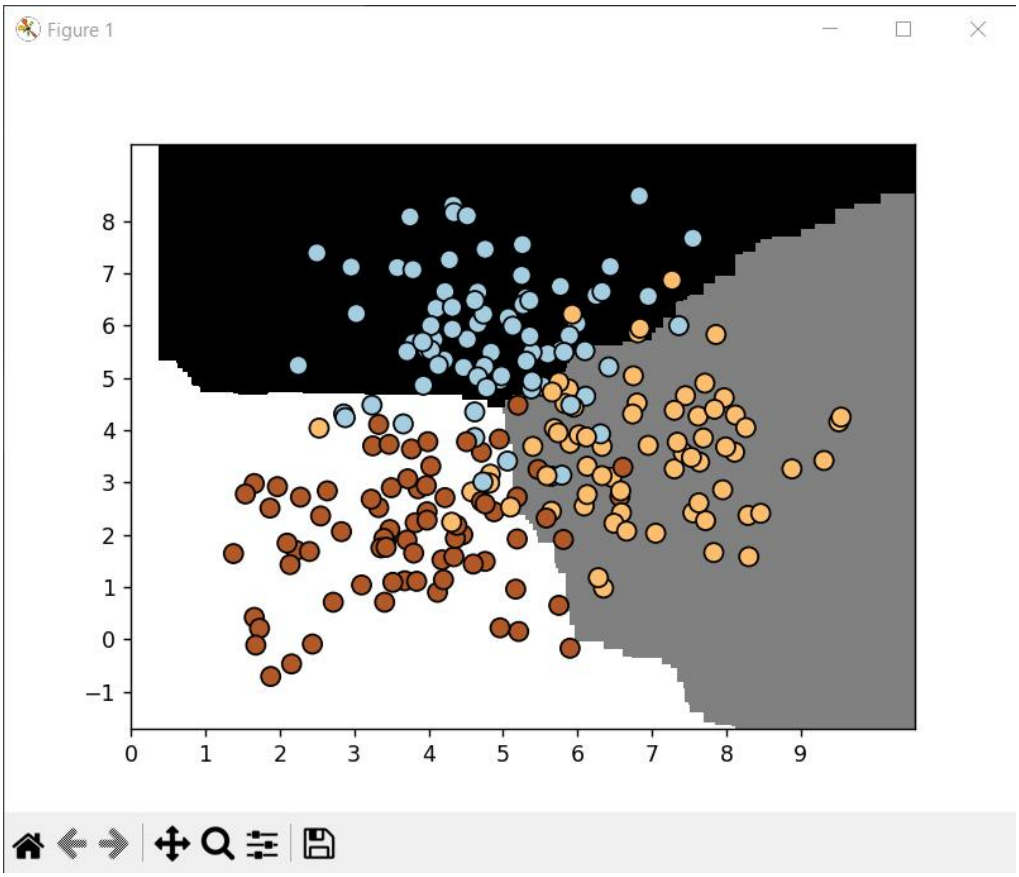


Рис.4.3.4. Візуалізація даних для другої метрики.

```
##### Searching optimal parameters for recall_weighted

Scores for the parameter grid:
mean_fit_time --> [0.10215716 0.10515394 0.09568548 0.1218071  0.13488817 0.02406363
 0.0512743  0.08313756 0.2359302 ]
std_fit_time --> [0.01634063 0.01275081 0.0117772  0.01987748 0.01143089 0.00204717
 0.00526707 0.00474051 0.06431838]
mean_score_time --> [0.00928531 0.00883722 0.00837336 0.00979056 0.01113195 0.00364566
 0.00435567 0.00920582 0.01994743]
std_score_time --> [0.00284224 0.00338163 0.0024329  0.00234516 0.00289396 0.00070282
 0.00052986 0.00140212 0.00483424]
param_max_depth --> [2 4 7 12 16 4 4 4 4]
param_n_estimators --> [100 100 100 100 100 25 50 100 250]
params --> [{'max_depth': 2, 'n_estimators': 100}, {'max_depth': 4, 'n_estimators': 100}, {'max_depth': 7,
'n_estimators': 100}, {'max_depth': 12, 'n_estimators': 100}, {'max_depth': 16, 'n_estimators': 100}, {'max
_depth': 4, 'n_estimators': 25}, {'max_depth': 4, 'n_estimators': 50}, {'max_depth': 4, 'n_estimators': 100
}, {'max_depth': 4, 'n_estimators': 250}]
split0_test_score --> [0.87407407 0.85185185 0.85185185 0.81481481 0.8        0.87407407
 0.84444444 0.85185185 0.85925926]
split1_test_score --> [0.86666667 0.85925926 0.87407407 0.87407407 0.85185185 0.85185185
 0.84444444 0.85925926 0.87407407]
split2_test_score --> [0.80740741 0.82222222 0.82222222 0.80740741 0.77037037 0.82962963
 0.82962963 0.82222222 0.82222222]
split3_test_score --> [0.81481481 0.8        0.8        0.8        0.79259259 0.79259259
 0.8        0.8        0.8        ]
split4_test_score --> [0.85185185 0.85185185 0.85925926 0.85185185 0.85925926 0.86666667
 0.85925926 0.85185185 0.85185185]
mean_test_score --> [0.84296296 0.83703704 0.84148148 0.82962963 0.81481481 0.84296296
 0.83555556 0.83703704 0.84148148]
std_test_score --> [0.02707506 0.02246778 0.02674884 0.02849687 0.03474382 0.02940657
 0.02009579 0.02246778 0.02674884]
rank_test_score --> [1 5 3 8 9 1 7 5 3]

Best parameters: {'max_depth': 2, 'n_estimators': 100}
```

Рис.4.3.5. Сітковий пошук для другої метрики.

```
#########################################
Classifier performance on training dataset
              precision    recall  f1-score   support

     Class-0       0.94      0.81      0.87        79
     Class-1       0.81      0.86      0.83        70
     Class-2       0.83      0.91      0.87        76

    accuracy                           0.86       225
   macro avg       0.86      0.86      0.86       225
weighted avg       0.86      0.86      0.86       225

#########################################
```

Рис.4.3.6. Оцінка класифікації для другої метрики.

Найкращим набором параметрів у обох випадках є max_depth = 2

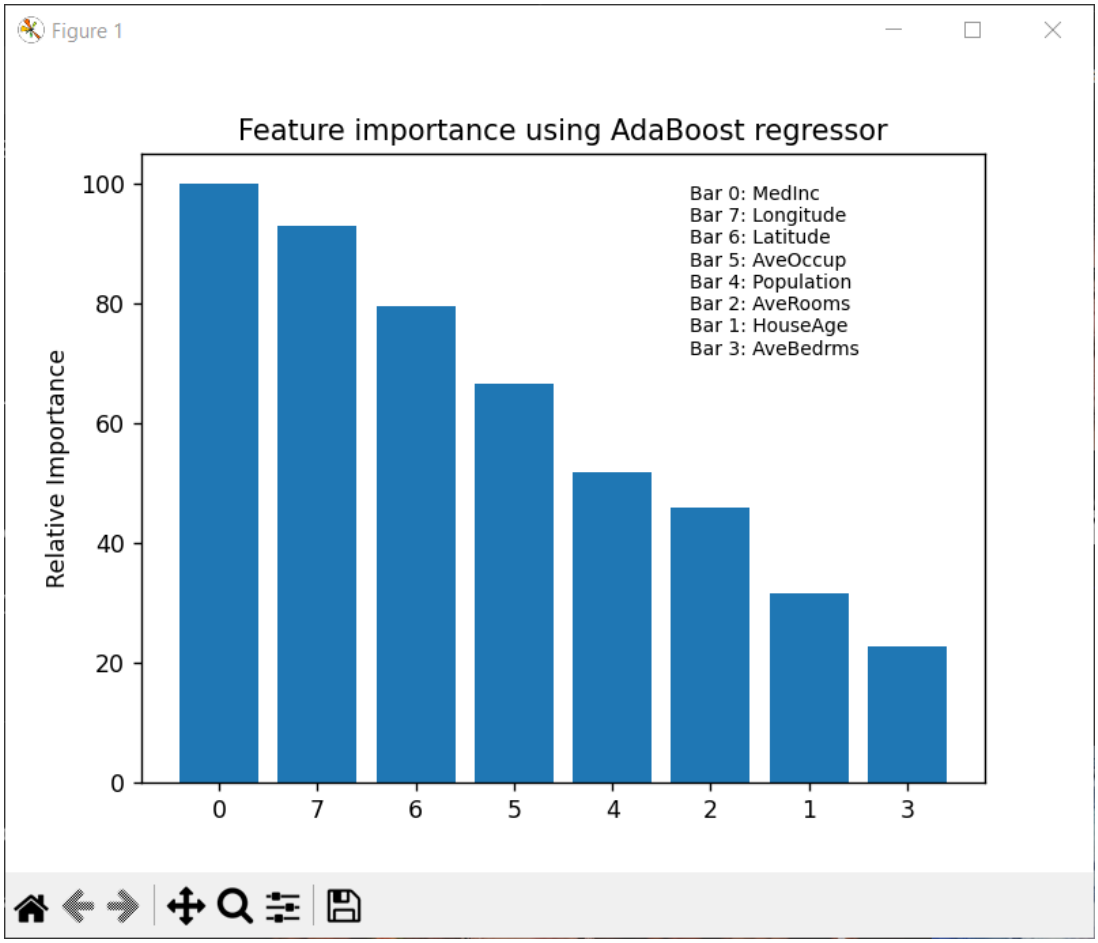
n_estimators = 100. Обидва випадки показали високі результати. Для параметрів, визначених як оптимальні, середня оцінка тесту для метрик 'precision_weighted' та 'recall_weighted' складає близько 0.85. Результати виявились стабільними для обох метрик, підтверджуючи точність моделі на різних наборах даних. Побудована модель класифікації має хорошу узагальнюючу здатність та може добре працювати на нових даних.

Завдання №4.4. Обчислення відносної важливості ознак.

Лістинг файлу LR_4_task_4.py:

```
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

# 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 = 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 X
pos = np.arange(index_sorted.shape[0]) + 0.5
```

```
# Побудова стовпчастої діаграми
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, index_sorted)
plt.text(0.65, 0.95, '\n'.join([f'Bar {i}: {feature_names[i]}' for i in
index_sorted]),
         transform=plt.gca().transAxes, fontsize=8, va='top')
plt.ylabel('Relative Importance')
plt.title('Feature importance using AdaBoost regressor')
plt.show()
```




```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_4.py

ADABOOST REGRESSOR
Mean squared error = 1.18
Explained variance score = 0.47

D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.4.1 – 4.4.2. Результат виконання завдання.

Згідно з отриманою діаграмою, ознаки, які мають найбільший вплив на ціни на нерухомість, це MedInc, Longtitude та Latitude (>=80%). AveOccup, Population, AveRooms мають середній вплив на ціни, їхня значимість знаходиться в діапазоні від 40% до 70%. Найменше впливають HouseAge і AveBedrms (<=30%).

Модель AdaBoostRegressor, за результатами, має середньоквадратичну помилку на рівні 1.18, що вказує на присутність помилок у прогнозуванні, проте вона в змозі пояснити приблизно 47% дисперсії даних. Модель має певний рівень пояснювальної здатності, але потребує покращення.

Завдання №4.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів.

Лістинг файлу LR_4_task_5.py:

```
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

# 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]))
```

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_5.py
Mean absolute error: 7.42
Predicted traffic: 26

D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.5.1. Результат виконання завдання.

Завдання №4.6. 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 файлу LR_4_task_6.py:

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_6.py

Predicted output:
 [1 2 2 0 2 0 2 1 0 1 1 2 1 0 2 2 1 0 0 1 0 2 1 1 2 2 0 0 1 2 1 2 1 0 2 2 1
 1 2 2 2 0 1 2 2 1 2 2 1 0 1 2 2 2 2 0 2 2 0 2 2 0 1 0 2 1 1 1 1 2 0 1 0 2
 0 0 1 2 2 0 0 2 2 2 2 0 0 0 2 2 2 1 2 0 2 0 2 2 0 0 1 1 1 1 2 2 2 2 0 1 1
 0 2 1 1 0 1 1 1 1 0 0 0 1 2 1 0 0 2 1 2 0 0 1 0 1 1 0 1 1 1 2 2 2 0 1 2 0
 2 2]

Score: 0.9066666666666666

Indices of selected features: 4, 7, 8, 12, 14, 17, 22

D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.6.1. Результат виконання завдання.

Predicted output містить передбачені вихідні значення, які є результатом застосування моделі для вхідних даних. Вони представлені у вигляді чисел від 0 до 2, що вказують на прогнозовані класи.

Значення Score – оцінка моделі. Значення 0.9066 означає, що модель досягла точності близько 90.67%.

Indices of selected features містить інформацію про індекси обраних ознак або функцій, які є важливими для прогнозування моделі.

Побудована модель демонструє високу точність передбачення, а також успішно ідентифікує важливі ознаки, що сприяє її ефективності у прогнозуванні результатів на основі вхідних даних.

Завдання №4.7. Пошук найближчих сусідів.

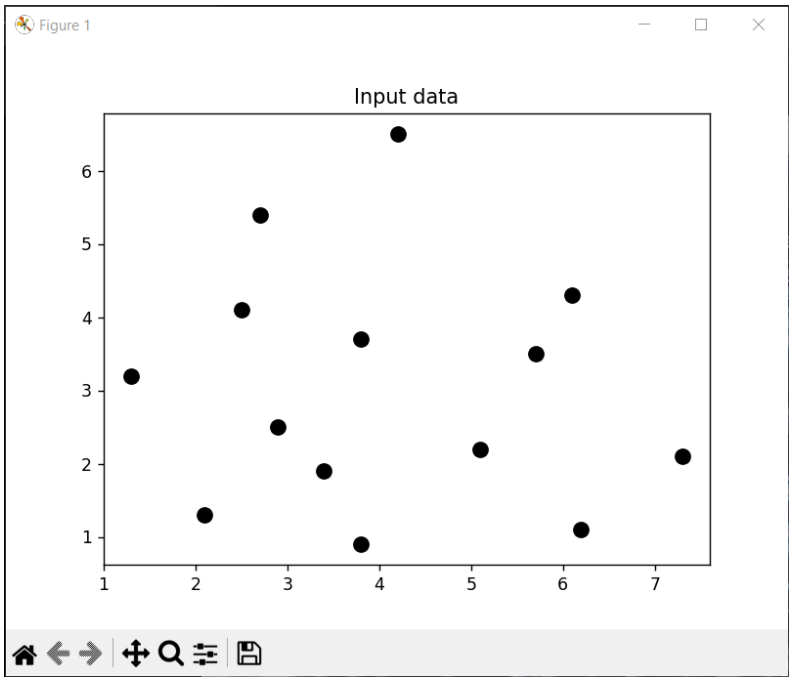
Для формування ефективних рекомендацій у рекомендаційних системах використовується поняття найближчих сусідів (nearest neighbours), суть якого полягає у знаходженні тих точок заданого набору, які розташовані на найближчих відстанях від зазначеної. Такий підхід часто застосовується для створення систем,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

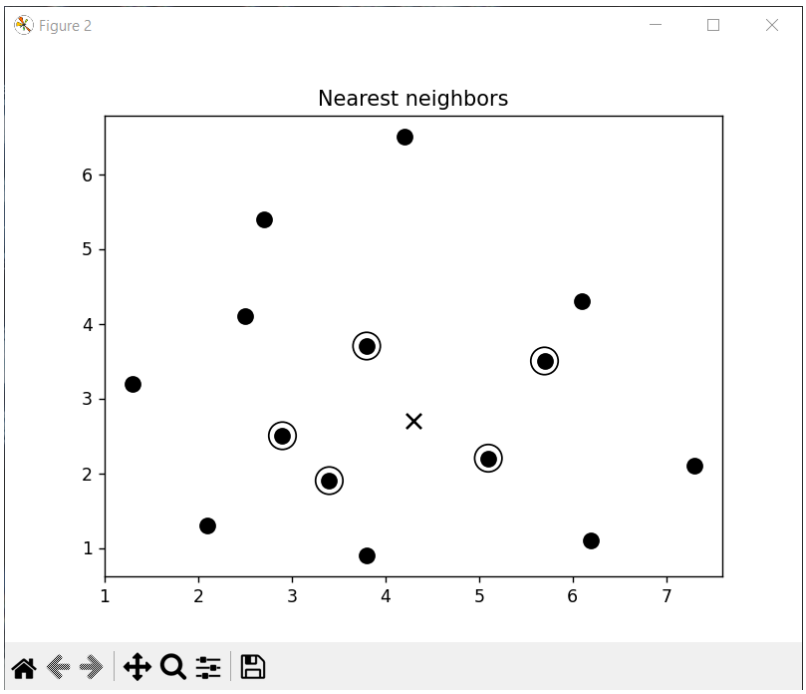
Здійсніть пошук найближчих сусідів заданої точки даних.

Лістинг файлу LR_4_task_7.py:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors

# Вхідні дані
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]])

# Кількість найближчих сусідів
k = 5

# Тестова точка даних
test_datapoint = [4.3, 2.7]

# Відображення вхідних даних на графіку
plt.figure()
plt.title('Input data')
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='black')

# Побудова моделі на основі методу k найближчих сусідів
knn_model = NearestNeighbors(n_neighbors=k, algorithm='ball_tree').fit(X)
distances, indices = knn_model.kneighbors([test_datapoint])
```

```
# Виведемо 'k' найближчих сусідів
print("\nK Nearest Neighbors:")
for rank, index in enumerate(indices[0][:k], start=1):
    print(str(rank) + " ==>", X[index])

# Візуалізація найближчих сусідів разом із тестовою точкою даних
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()
```





```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_7.py

K Nearest Neighbors:
1 ==> [5.1 2.2]
2 ==> [3.8 3.7]
3 ==> [3.4 1.9]
4 ==> [2.9 2.5]
5 ==> [5.7 3.5]

D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.7.1 – 4.7.3. Результат виконання завдання.

На першому графіку відображено вхідні дані, на другому – обрана точка і 5 найближчих до неї сусідів, а в терміналі зазначено їхні координати.

Завдання №4.8. Створити класифікатор методом k найближчих сусідів.

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

Лістинг файлу LR_4_task_8.py:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors, datasets

# Завантаження вхідних даних
input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1].astype(int)

# Відображення вхідних даних на графіку
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')

# Кількість найближчих сусідів
num_neighbors = 12
step_size = 0.01

# Створення класифікатора на основі методу k найближчих сусідів
classifier = neighbors.KNeighborsClassifier(num_neighbors, weights='distance')

# Навчання моделі на основі методу k найближчих сусідів
classifier.fit(X, y)

# Створення сітки для відображення меж на графіку
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))
```

```
# Виконання класифікатора на всіх точках сітки
output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])

# Візуалізація передбачуваного результату
output = output.reshape(x_values.shape)
plt.figure()
plt.pcolormesh(x_values, y_values, output, cmap=cm.Paired)

# Накладання навчальних точок на карту
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('K Nearest Neighbors classifier model boundaries')

# Тестування вхідної точки даних
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')

# Вилучення K найближчих сусідів
_, indices = classifier.kneighbors([test_datapoint])
indices = indices.astype(int)[0]

# Відображення K найближчих сусідів на графіку
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()
```

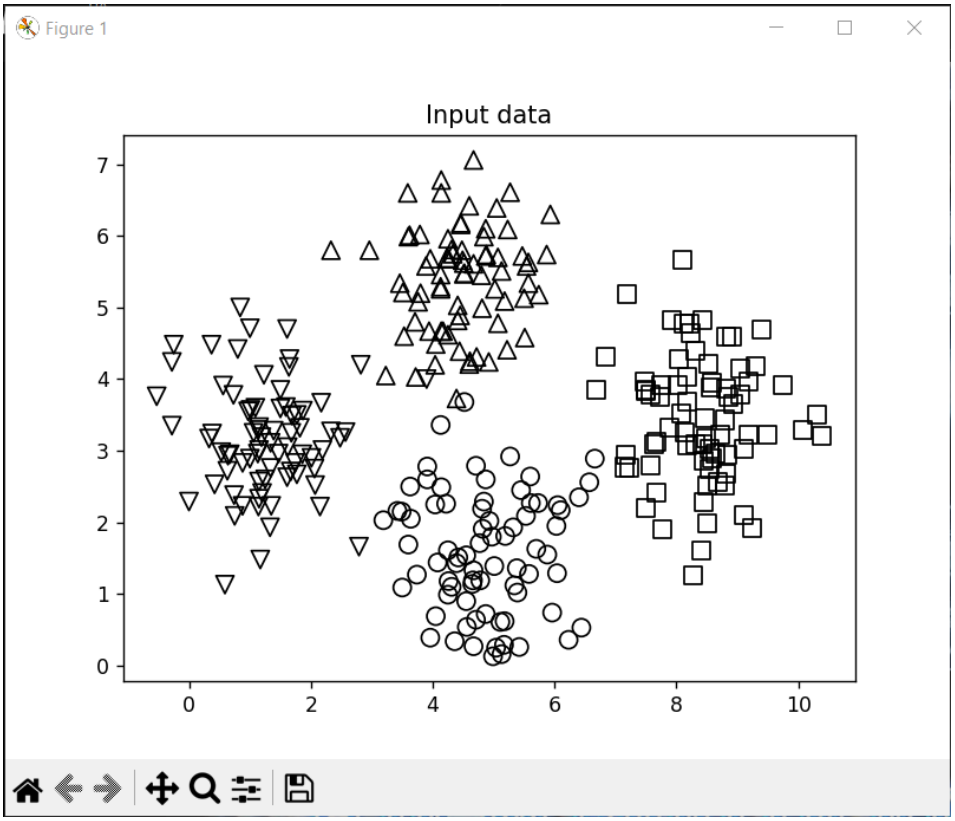


Рис.4.8.1. Вхідні дані.

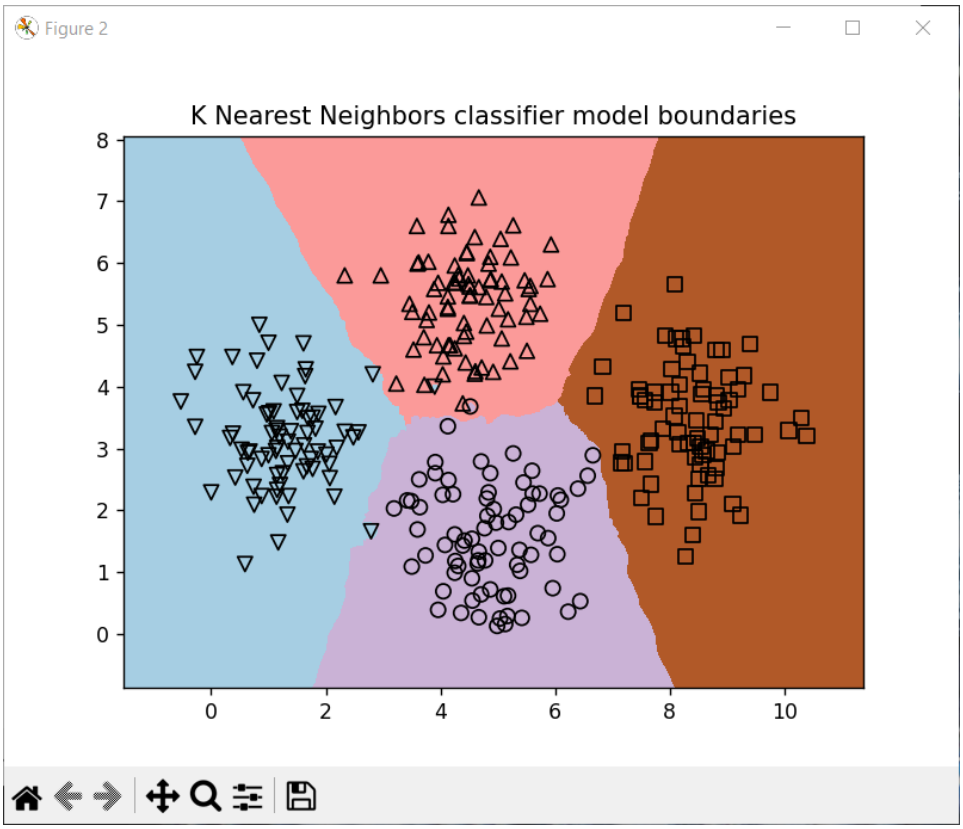


Рис.4.8.2. Межі моделі класифікатора.

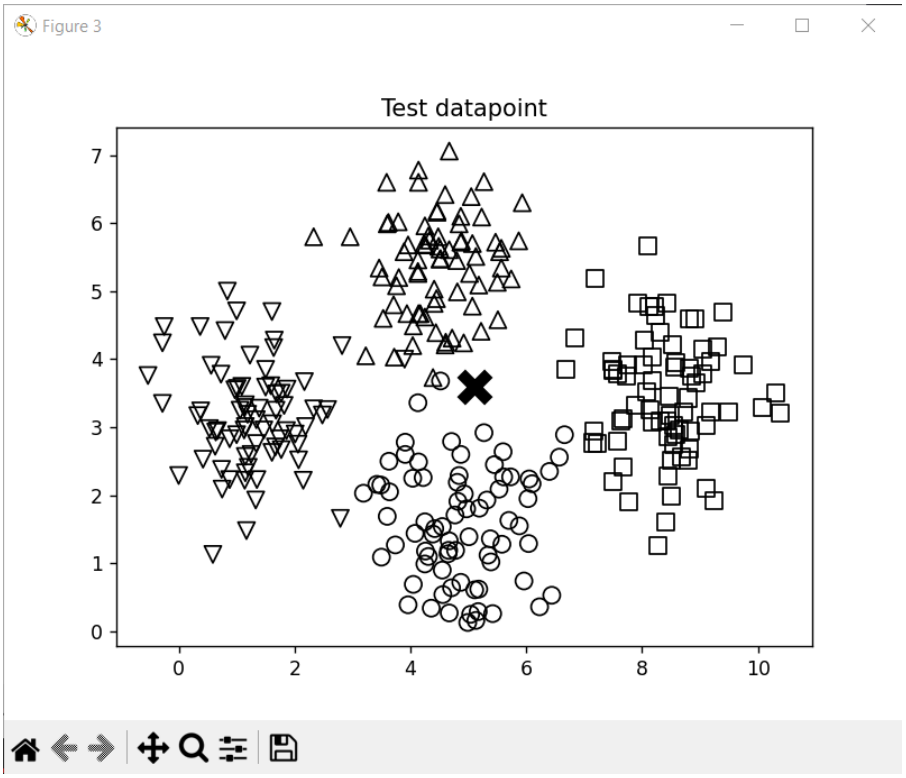


Рис.4.8.3. Позначення тестової точки.

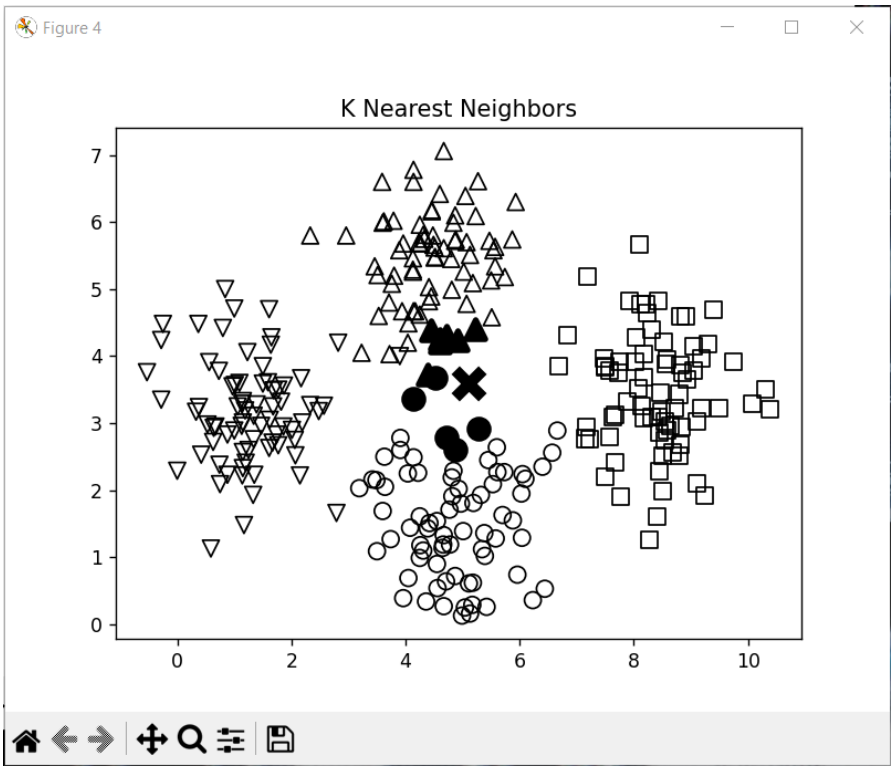


Рис.4.8.4. Найближчі сусіди тестової точки.

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_8.py
Predicted output: 1

D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.8.5. Результат виконання завдання.

На першій діаграмі показано вхідні дані, другій – межі моделі класифікатора з визначеними класами, третій – тестова точка, на четвертій – найближчі сусіди до тестової точки. У терміналі було виведено, що точка належить до класу 1.

Завдання №4.9. Обчислення оцінок подібності.

Лістинг файлу LR_4_task_9.py:

```
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
metricto be used')
    return parser


# Обчислення оцінки евклідова відстані між користувачами user1 та user2
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')

    # Фільми, оцінені обома користувачами, user1 та user2
    common_movies = {}
    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1

    # За відсутності фільмів, оцінених обома користувачами, оцінка приймається рі-
вною 0
    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)))


# Обчислення кореляційної оцінки Пірсона між користувачем1 і користувачем2
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')
```

```
        # Фільми, оцінені обома користувачами, user1 та user2
    common_movies = {}
    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1

    num_ratings = len(common_movies)
    # За відсутності фільмів, оцінених обома користувачами, оцінка приймається рі-
вною 0
    if num_ratings == 0:
        return 0

    # Обчислення суми рейтингових оцінок усіх фільмів, оцінених обома користувача-
ми
    user1_sum = np.sum([dataset[user1][item] for item in common_movies])
    user2_sum = np.sum([dataset[user2][item] for item in common_movies])

    # Обчислення Суми квадратів рейтингових оцінок всіх фільмів, оцінених обома
користувачами
    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])

    # Обчислення суми творів рейтингових оцінок всіх фільмів, оцінених обома кори-
стувачами
    sum_of_products = np.sum([dataset[user1][item] * dataset[user2][item] for item
                              in common_movies])
    # Обчислення коефіцієнта кореляції Пірсона
    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))
```

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy"
--score-type Euclidean

Euclidean score:
0.585786437626905

D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy"
--score-type Pearson

Pearson score:
0.9909924304103233

D:\course-4\semester-1\ai\lab4_project>
```

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith" --user2 "Brenda Peterson"
 --score-type Euclidean

Euclidean score:
0.1424339656566283

D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith" --user2 "Brenda Peterson"
 --score-type Pearson

Pearson score:
-0.7236759610155113

D:\course-4\semester-1\ai\lab4_project>
```

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller"
--score-type Euclidean

Euclidean score:
0.30383243470068705

D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller"
--score-type Pearson

Pearson score:
0.7587869106393281

D:\course-4\semester-1\ai\lab4_project>
```

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel"
--score-type Euclidean

Euclidean score:
0.2857142857142857

D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel"
--score-type Pearson

Pearson score:
0

D:\course-4\semester-1\ai\lab4_project>
```

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith"
 --user2 "Clarissa Jackson" --score-type Euclidean

Euclidean score:
0.28989794855663564

D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith"
 --user2 "Clarissa Jackson" --score-type Pearson

Pearson score:
0.6944217062199275

D:\course-4\semester-1\ai\lab4_project>
```

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith"
 --user2 "Adam Cohen" --score-type Euclidean

Euclidean score:
0.38742588672279304

D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith"
 --user2 "Adam Cohen" --score-type Pearson

Pearson score:
0.9081082718950217

D:\course-4\semester-1\ai\lab4_project>
```

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith"
 --user2 "Chris Duncan" --score-type Euclidean

Euclidean score:
0.38742588672279304

D:\course-4\semester-1\ai\lab4_project>python LR_4_task_9.py --user1 "David Smith"
 --user2 "Chris Duncan" --score-type Pearson

Pearson score:
1.0

D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.9.1 – 4.9.7. Результат виконання завдання.

Евклідова оцінка вимірює схожість об’єктів шляхом вимірювання відстані між їхніми атрибутами. Чим менша оцінка, тим більша відстань, а значить менша подібність між об’єктами. Коефіцієнт кореляції Пірсона обчислюється як коваріація між двома змінними, поділена на добуток їх стандартних відхилень, і може приймати значення від -1 до +1, де значення 1 вказує на повну позитивну лінійну залежність, 0 - на відсутність кореляції, а -1 - на повну негативну лінійну залежність між змінними.

Завдання №4.10. Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації.

Лістинг файлу LR_4_task_10.py:

```
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 to
```

```
the 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')
        # Обчислення оцінки подібності за Пірсоном між
        # вказаним користувачем та всіма іншими
        # користувачами в наборі даних
    scores = np.array([[x, pearson_score(dataset, user, x)] for x in dataset if x
!= user])

    # Сортування оцінок за спаданням
    scores_sorted = np.argsort(scores[:, 1])[::-1]

    # Вилучення оцінок перших 'num_users' користувачів
    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))
```

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_10.py --user "Bill Duffy"

Users similar to Bill Duffy:

User                    Similarity score
-----------------------------------------
David Smith              0.99
Samuel Miller            0.88
Adam Cohen               0.86

D:\course-4\semester-1\ai\lab4_project>python LR_4_task_10.py --user "Clarissa Jackson"

Users similar to Clarissa Jackson:

User                    Similarity score
-----------------------------------------
Chris Duncan             1.0
Bill Duffy               0.83
Samuel Miller            0.73

D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.10.1. Результат виконання завдання.

У результаті знайдено користувачів зі схожими уподобаннями та їхні оцінки

за допомогою коефіцієнта кореляції Пірсона. Таким чином, уподобання Bill Duffy і David Smith майже ідентичні, а Clarissa Jackson і Chris Duncan – повністю співпадають.

Завдання №4.11. 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 файлу LR_4_task_11.py:

```
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


# Отримання рекомендації щодо фільмів для вказаного користувача
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']

    # Генерація рейтингів фільмів за допомогою їх нормалізації
    movie_scores = np.array([[score / similarity_scores[item], item] for item,
score in overall_scores.items()])

    # Сортування за спаданням
    movie_scores = movie_scores[np.argsort(movie_scores[:, 0])[::-1]]
```

```
    # Вилучення рекомендацій фільмів
    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)
```

```
D:\course-4\semester-1\ai\lab4_project>python LR_4_task_11.py --user "Chris Duncan"

Movie recommendations for Chris Duncan:
1. Vertigo
2. Scarface
3. Goodfellas
4. Roman Holiday

D:\course-4\semester-1\ai\lab4_project>python LR_4_task_11.py --user "Julie Hammel"

Movie recommendations for Julie Hammel:
1. The Apartment
2. Vertigo
3. Raging Bull

D:\course-4\semester-1\ai\lab4_project>
```

Рис.4.11.1. Результат виконання завдання.

Написана модель аналізує вподобання заданого користувача та решти користувачів. Якщо подібність вподобань більше 0, модель обирає не оцінені першим користувачем фільми і сортує їх, таким чином створюючи список рекомендацій. Як можна побачити, було успішно сформовано списки для різних користувачів.

Висновки: в ході виконання лабораторної роботи було досліджено методи ансамблів у машинному навчанні та створено рекомендаційні системи, використовуючи спеціалізовані бібліотеки та мову програмування Python.