ЛАБОРАТОРНА РОБОТА № 4

Тема: **«ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ»**

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

Хід роботи

Посилання на програмнй код на Github: https://github.com/NightSDay/AI-2023/tree/main/AI-subject/L-4

Завдання 2.1. **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', edgecol-
ors='black', linewidth=1, marker='s')
    plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white', edgecol-
ors='black', linewidth=1, marker='o')
    plt.scatter(class_2[:, 0], class_2[:, 1], s=75, facecolors='white', edgecol-
ors='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, ran-
dom_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), tar-
get_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()
```

Результат виконання програми:

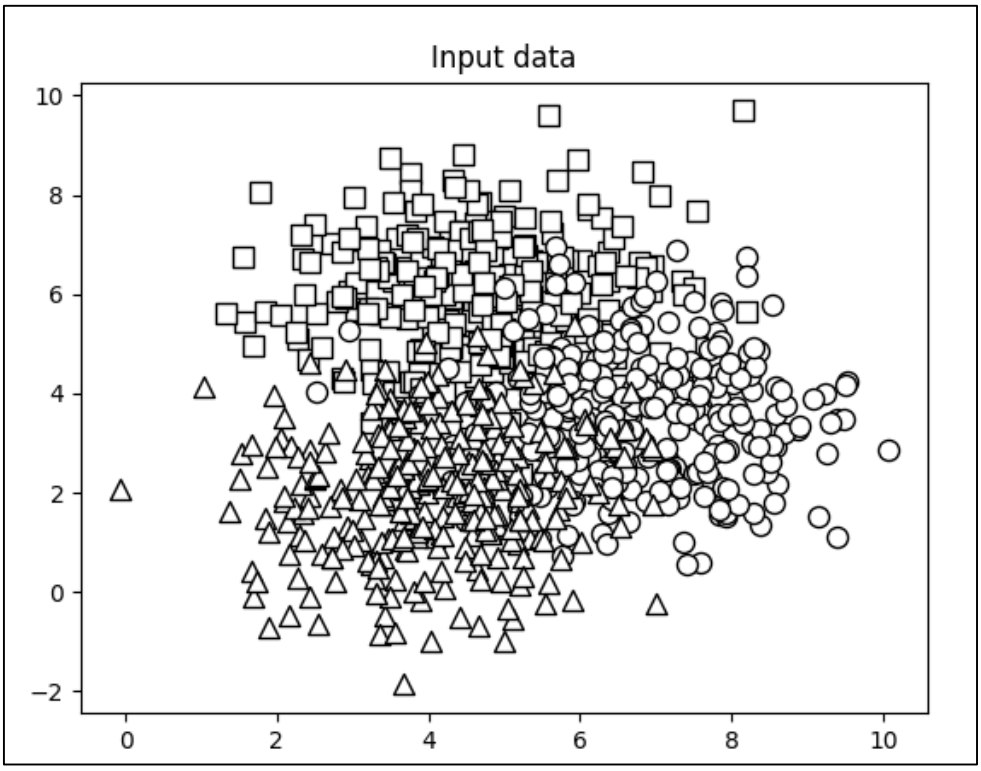


Рис. 1.1. Результат виконання програми (Результат розподілення даних)

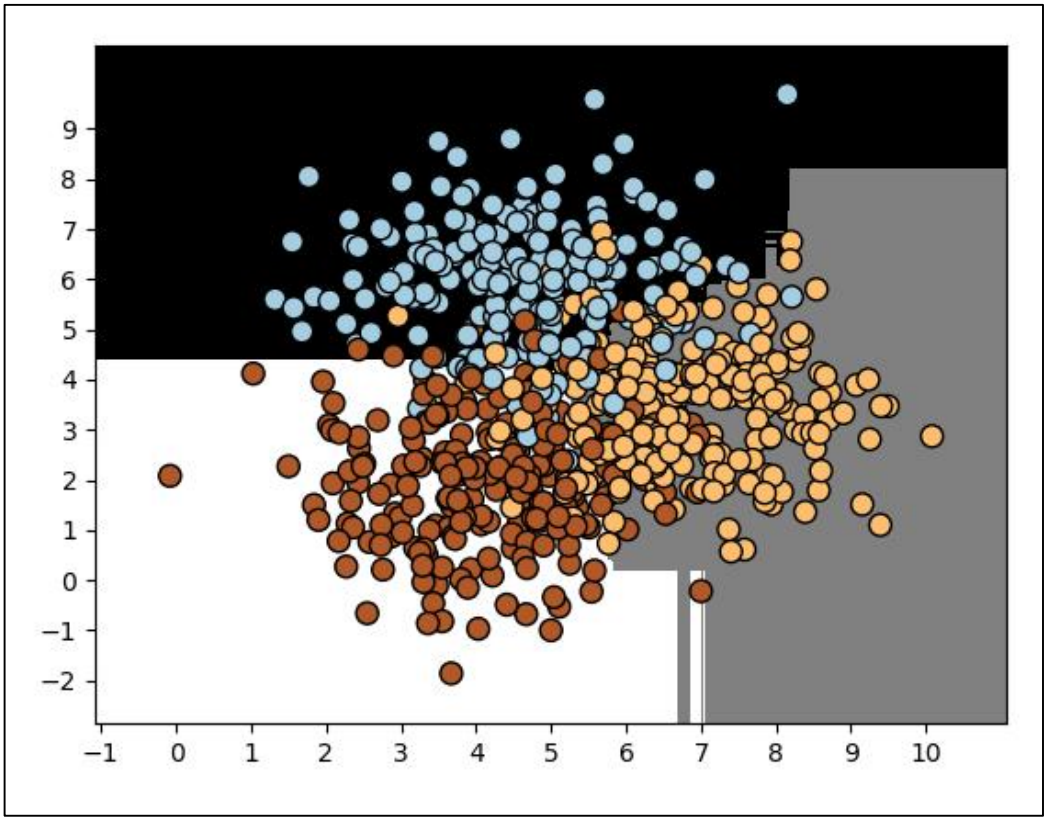


Рис. 1.2. Результат виконання програми

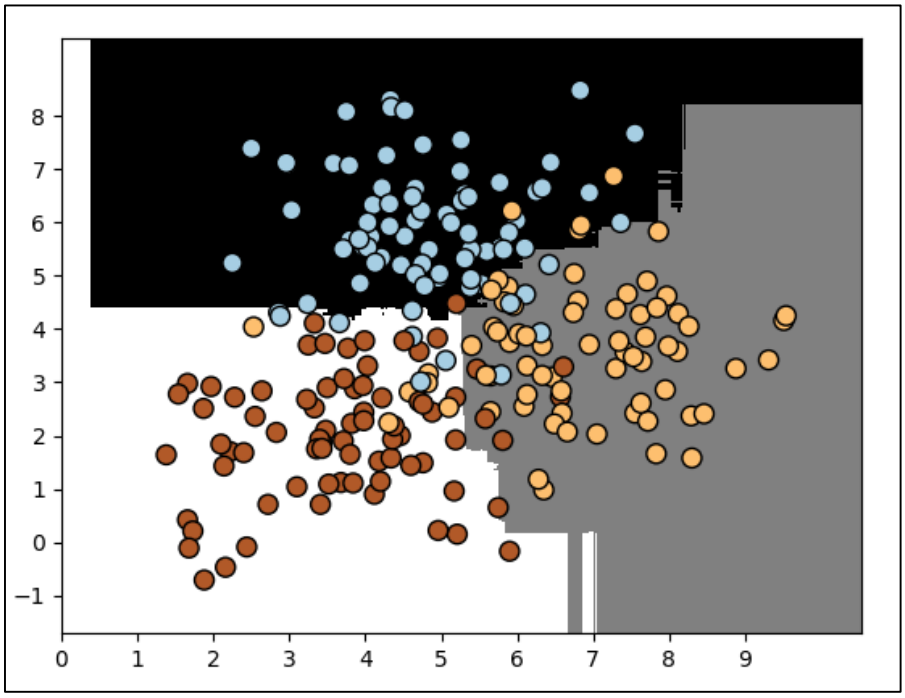


Рис. 1.3. Результат виконання програми

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

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

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


C:\__My\__7-SEM\AI\AI_Labs_Git\ipz-20-2_bubenko\lab4>
```

Рис. 1.4. Результат виконання програми (характеристики роботи методу випадкових лісів)

Тепер виконайте той самий код, запросивши створення класифікатора на основі гранично випадкового лісу за допомогою прапорця erf вхідного аргументу

Результат виконання програми:

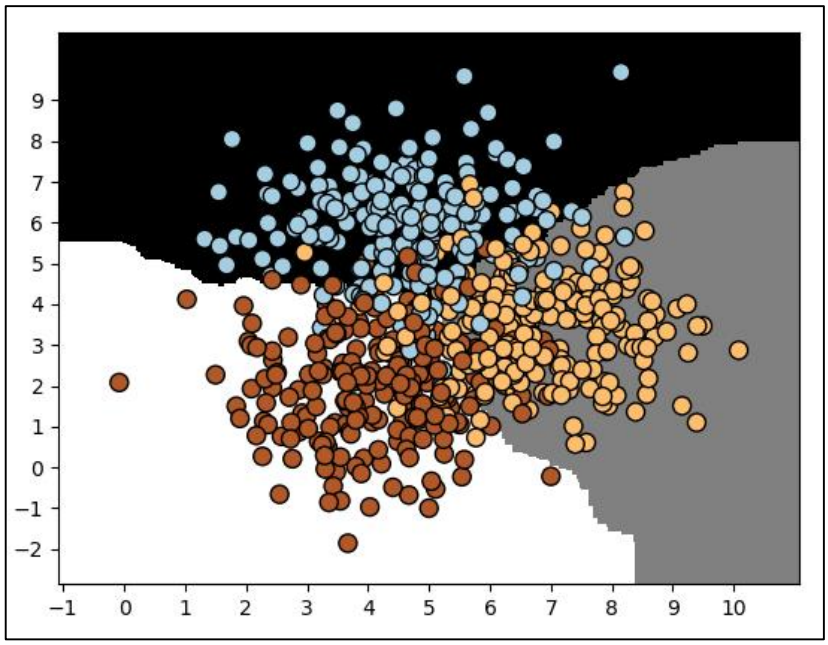


Рис. 1.5. Результат виконання програми

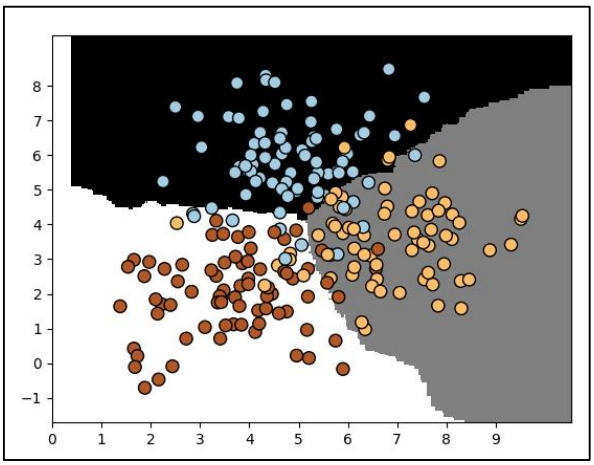


Рис. 1.6. Результат виконання програми

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

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

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

Рис. 1.7. Результат виконання програми (характеристики роботи методу гранично випадкових лісів)

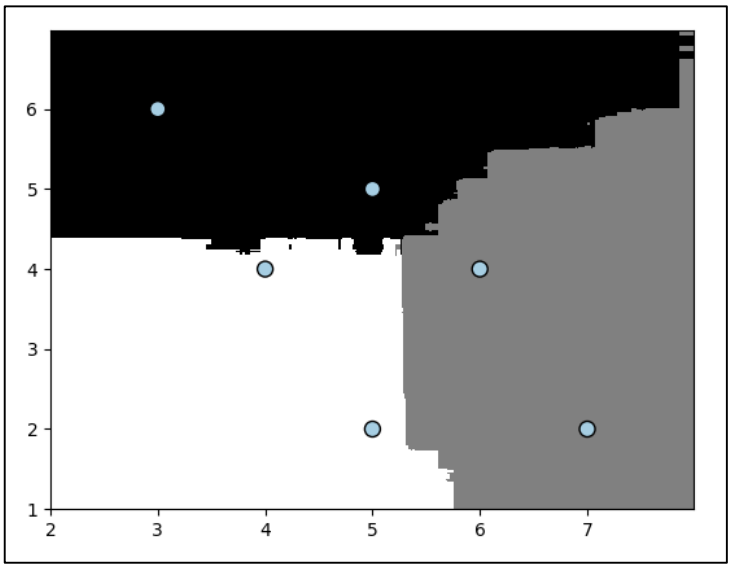


Рис. 1.8. Візуалізація можливих класів точок (метод випадкових лісів)

```
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
```

Рис. 1.9. Дані про можливі класи (метод випадкових лісів)

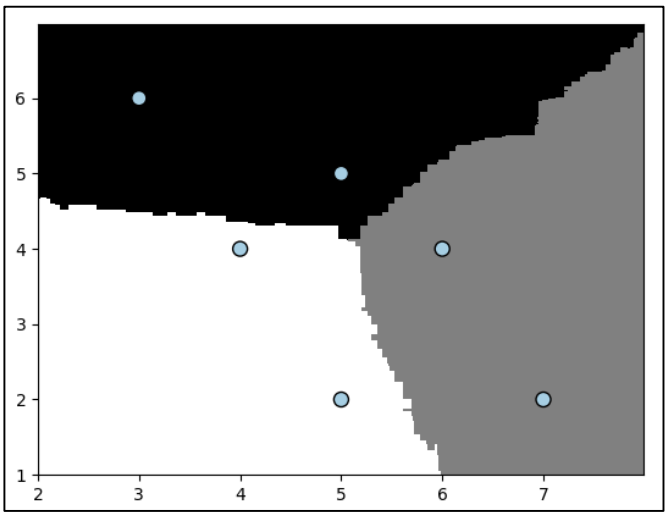


Рис. 1.10. Візуалізація можливих класів точок (метод гранично випадкових лісів)

```
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
```

Рис. 1.11. Дані про можливі класи (метод гранично випадкових лісів)

Висновки, щодо результатів класифікації на основі випадкових дерев та гранично випадкових лісів

Методи випадкових дерев та гранично випадкових лісів належать до найбільш ефективних методів класифікації даних. У нашому випадку обидва методи виявилися однаково ефективними. Проте, гранично випадкові ліси можуть бути кращим вибором у випадках, коли важлива швидкість алгоритму та існує ризик перенавчання. Крім того, гранично випадкові ліси відзначаються високою ефективністю при роботі з великими обсягами даних та здатністю ефективно враховувати кореляції між ознаками.

Завдання 2.2. **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', edgecol-
ors='black', linewidth=1, marker='o')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ran-
dom_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), tar-
get_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, ze-
ro_division=1))
print("#" * 40 + "\n")

plt.show()
```

Результат виконання програми:

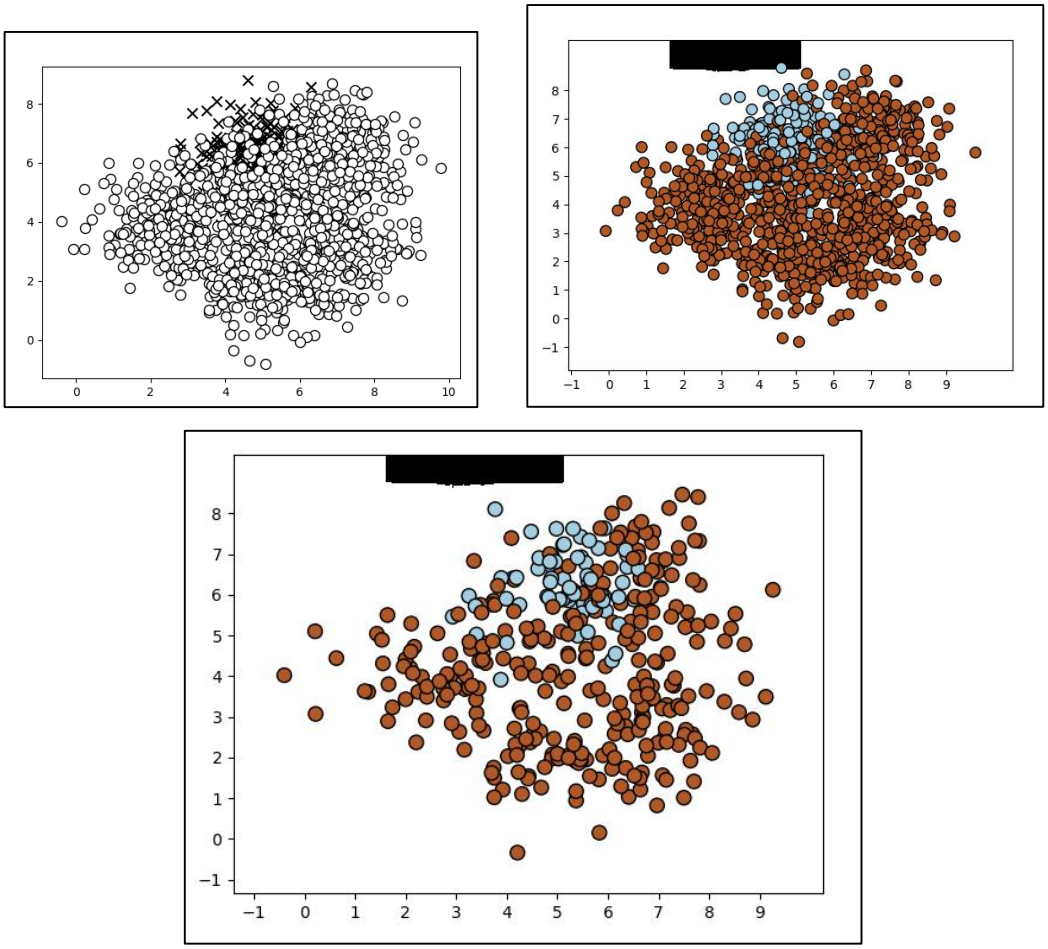
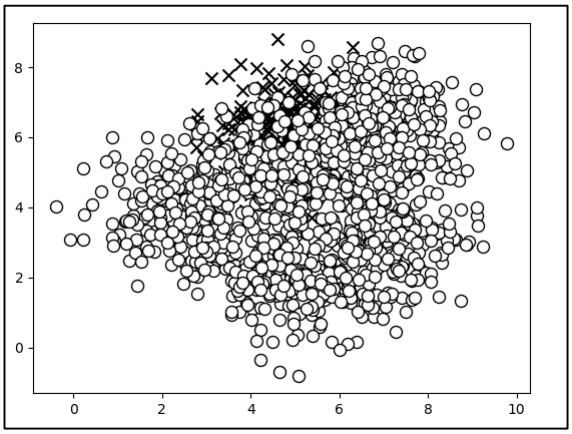
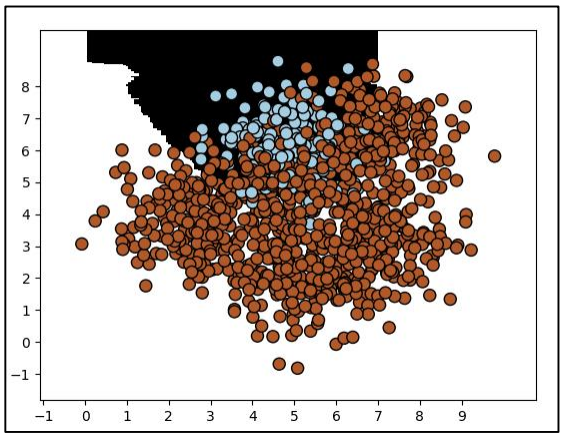


Рис. 2.1. – 2.3. Розподілення незбалансованих даних (вхідні, навчальні, тестові)

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

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

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

Рис. 2.4. Характеристики незбалансованої класифікації





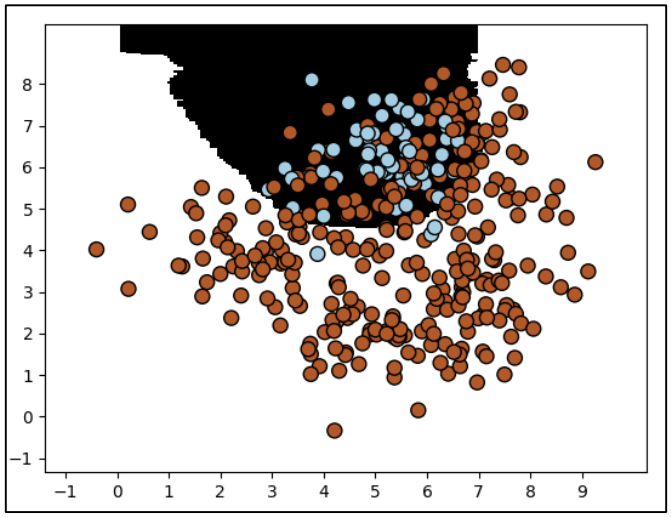


Рис. 2.4 – 2.6. Розподілення збалансованих даних (вхідні, навчальні, тестові)

```
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

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

Рис. 2.7. Характеристики збалансованої класифікації

Висновки, щодо обробку з урахуванням дисбалансу класів

Після вирівнювання класів було помічено значне підвищення точності для обох класів (Class-0 та Class-1) під час класифікації на тренувальному та тестовому наборах даних. Це підтверджує важливість збалансування даних у випадках, де спостерігається дисбаланс між класами. Таким чином, збалансований набір даних сприяє здатності моделі краще розпізнавати та враховувати обидва класи, уникнувши переваги якогось одного класу.

Завдання 2.3. 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 програми:

```
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, ran-
dom_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), parame-
ter_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_test_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_test_pred, target_names=class_names))
    print("#"*40 + "\n")

    visualize_classifier(classifier, X_test, Y_test)
```

Результат виконання програми:

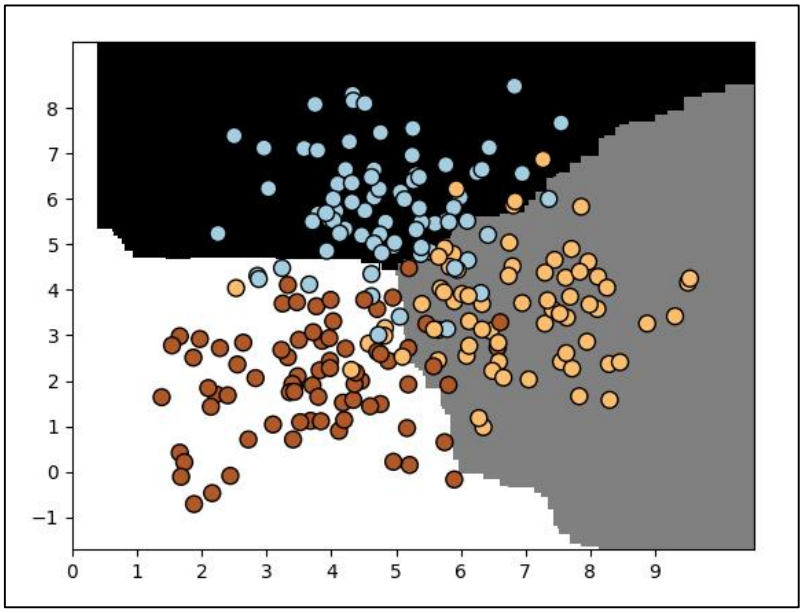


Рис. 3.1. Візуалізація класифікації даних зі сітковим пошуком (для першої метрики)

```
#### Searching optimal parameters for precision_weighted

Scores across the parameter grid:
mean_fit_time --> [0.08953505 0.10493183 0.09484735 0.11568713 0.13380332 0.02295299
 0.04369054 0.08798046 0.21931872]
std_fit_time --> [5.88846367e-03 1.46102653e-02 2.52016275e-03 1.79694899e-03
 5.35985443e-03 1.76844239e-05 7.18660763e-04 1.70010799e-03
 6.25572952e-03]
mean_score_time --> [0.00933776 0.00976596 0.00878353 0.00997658 0.0097754  0.0035759
 0.00518613 0.0085073  0.01835194]
std_score_time --> [0.00079108 0.00147432 0.0004046  0.00064273 0.00041894 0.0007917
 0.00040011 0.00044478 0.00047357]
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]

Highest scoring parameter set: {'max_depth': 2, 'n_estimators': 100}
```

Рис. 3.2. Отримання даних процесу класифікації (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. 3.3. Характеристика класифікації зі сітковим пошуком (для першої метрики)

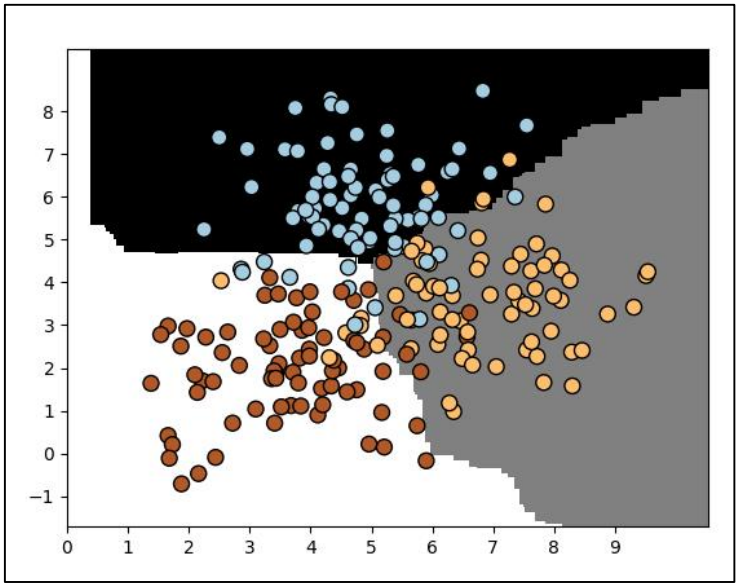


Рис. 3.4. Візуалізація класифікації даних зі сітковим пошуком (для другої метрики)

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

Scores across the parameter grid:
mean_fit_time --> [0.09782572 0.10768685 0.09643388 0.11707201 0.13345623 0.02276673
 0.04529462 0.0897006  0.22342186]
std_fit_time --> [0.01031181 0.01823137 0.00335238 0.00264437 0.00277199 0.00059326
 0.00101618 0.00369973 0.00440237]
mean_score_time --> [0.00984473 0.01017342 0.00937338 0.01008368 0.01056452 0.00379801
 0.00458121 0.00838614 0.01913509]
std_score_time --> [0.00117474 0.00232229 0.0005051  0.00049446 0.00135725 0.00074924
 0.00046926 0.00079371 0.00040887]
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]

Highest scoring parameter set: {'max_depth': 2, 'n_estimators': 100}
##########################################
```

Рис. 3.5. Отримання даних процесу класифікації (для другої метрики)

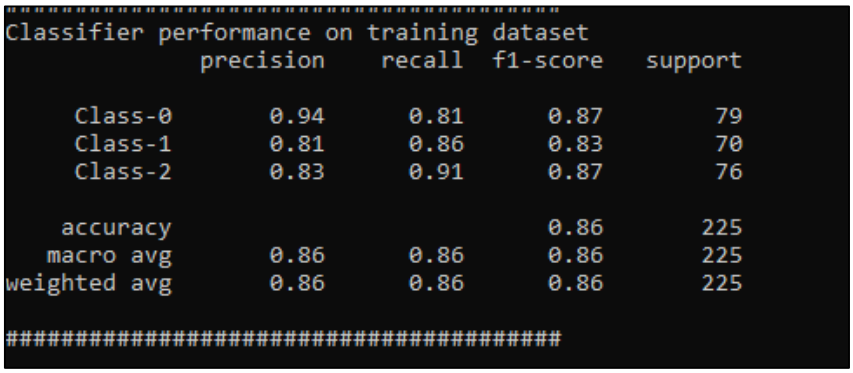


Рис. 3.6. Характеристика класифікації зі сітковим пошуком (для другої метрики)

Висновки щодо знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

Результати гіперпараметричного пошуку вказують на те, що оптимальні налаштування моделі, такі як max_depth=2 та n_estimators=100, є найефективнішими для обох метрик — precision_weighted і recall_weighted. Ці конфігурації сприяли високій якості класифікації, що в свою чергу привело до підвищення якості результатів у показниках precision та recall.

Завдання 2.4. **Обчислення відносної важливості ознак**

Так як метод load_boston був видалений з бібліотеки scikit-learn, використаємо дані про ціни на житло у Каліфорнії.

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

# 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, ran-
dom_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("ADABOOST 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 in-
dex_sorted]),
         transform=plt.gca().transAxes, fontsize=8, va='top')
plt.ylabel('Relative Importance')
plt.title('Feature importance using AdaBoost regressor')
plt.show()
```

Результат виконання програми:

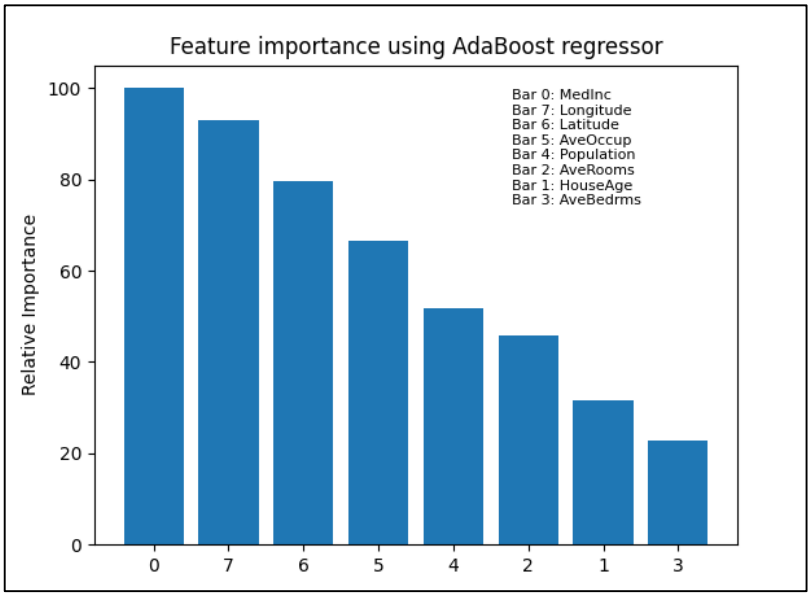


Рис. 4.1. Діаграма важливості ознак

```
E:\AI-subject\L-4\tasks>python LR_4_task_4.py
ADABOOST REGRESSOR
Mean squared error = 1.18
Explained variance score = 0.47

E:\AI-subject\L-4\tasks>
```

Рис. 4.2. Результат виконання програми

Висновки щодо відносної важливості ознак

Найбільш вагомими для моделі є MedInc, Longitude та Latitude, що свідчить про їх високий вплив на прогнозування. Додаткові характеристики, такі як Population, AveOccup і AveRooms, також вносять значний внесок у прогнозування, хоча їх вплив менший порівняно з першими трьома. Найменш значущими ознаками є AveBedrms та HouseAge, оскільки їх важливість найнижча серед усіх параметрів, які досліджувалися. У контексті цього набору даних, вплив Latitude та Longitude відображає нормальні географічні та просторові аспекти, що можуть суттєво впливати на вартість нерухомості в Каліфорнії. Помилка середнього квадрата становить 1.18, що вказує на відносно невелику похибку прогнозування. Проте, показник поясненої дисперсії на рівні 47% свідчить про обмежену здатність моделі у врахуванні варіацій вихідних даних.

Завдання 2.5. **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.

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

# 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, ran-
dom_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:

```
E:\AI-subject\L-4\tasks>python LR_4_task_5.py
Mean absolute error: 7.42
Predicted traffic: 26

E:\AI-subject\L-4\tasks>
```

Рис. 5. Результат виконання програми

Завдання 2.6. **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

# Генерація даних
X, y = _samples_generator.make_classification(n_samples=150,
                                              n_features=25, n_classes=3,
n_informative=6,
                                              n_redundant=0, random_state=7)

# Вибір k найважливіших ознак
k best selector = SelectKBest(f regression, k=9)

# Ініціалізація класифікатора на основі гранично випадкового лісу
classifier = ExtraTreesClassifier(n_estimators=60, max_depth=4)

# Створення конвеєра
processor_pipeline = Pipeline([('selector', k_best_selector), ('erf',
classifier)])

# Встановлення параметрів
processor_pipeline.set_params(selector__k=7, erf__n_estimators=30)

# Навчання конвеєра
processor_pipeline.fit(X, y)

# Прогнозування результатів для вхідних даних
output = processor_pipeline.predict(X)
print("\nPredicted output:\n", output)

# Виведення оцінки
print("\nScore:", processor_pipeline.score(X, y))

# Виведення ознак, відібраних селектором конвеєра
status = processor_pipeline.named_steps['selector'].get_support()

# Вилучення та виведення індексів обраних ознак
selected = [i for i, x in enumerate(status) if x]
print("\nIndices of selected features:", ', '.join([str(x) for x in selected]))
```

Результат виконання програми:

```
E:\AI-subject\L-4\tasks>python LR_4_task_6.py

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

Score: 0.8866666666666667

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

E:\AI-subject\L-4\tasks>
```

Рис. 6. Результат виконання програми

Висновки щодо результатів створення навчального конвеєра

Predicted output показує передбачені класи для тестового набору даних. Наприклад, перший запис може мати передбачений клас 1, що інтерпретується як приналежність цього запису до класу 1 за даними моделі. Score представляє метрику оцінки моделі. У цьому випадку, ймовірно, мова йде про точність моделі, яка складає 86%. Indices of selected features: 4, 7, 8, 12, 14, 17, 22 вказують на індекси ознак, які були відібрані моделлю як ключові для прийняття рішення. Отже, отримані результати свідчать про ефективність використаної моделі, оскільки вона продемонструвала високу точність на тестовому наборі даних, досягнувши 86%.

Завдання 2.7. **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

# 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()
```

Результат виконання програми:

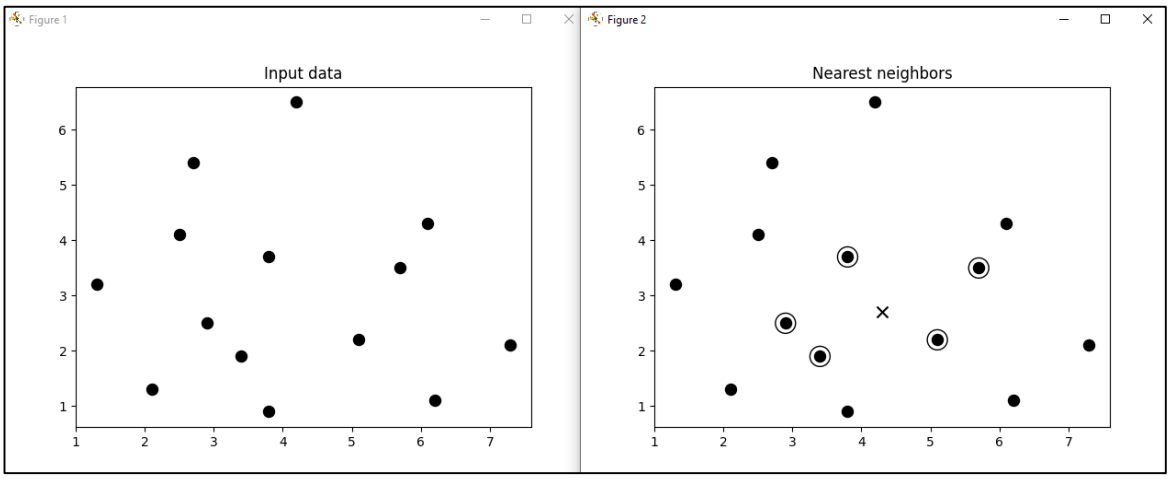


Рис. 7.1. Вхідні дані та пошук найближчих сусідів

```
E:\AI-subject\L-4\tasks>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]

E:\AI-subject\L-4\tasks>
```

Рис. 7.2. Інформація про найближчих сусідів

Висновок щодо результатів пошуку найближчих сусідів

На лівому графіку відображені вхідні дані, тоді як на правому графіку представлені найближчі сусіди, а їх координати виведені в терміналі. За результатами вибору k найближчих сусідів (у цьому випадку, k=5) для тестової точки можна зазначити, що п'ять найближчих точок навчального набору мають найбільш близькі значення до тестової точки з координатами [4.3, 2.7]. Це свідчить про те, що ці п'ять точок можуть мати схожі характеристики з тестовою точкою.

Завдання 2.8. **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

# 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

# Розмір кроку сітки візуалізації
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, face-
colors='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 програми:

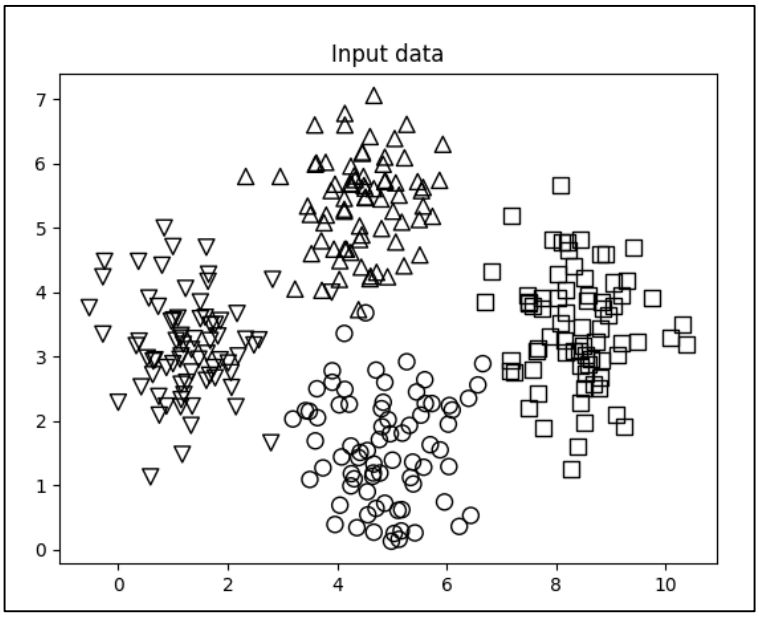


Рис. 8.1. Вхідні дані

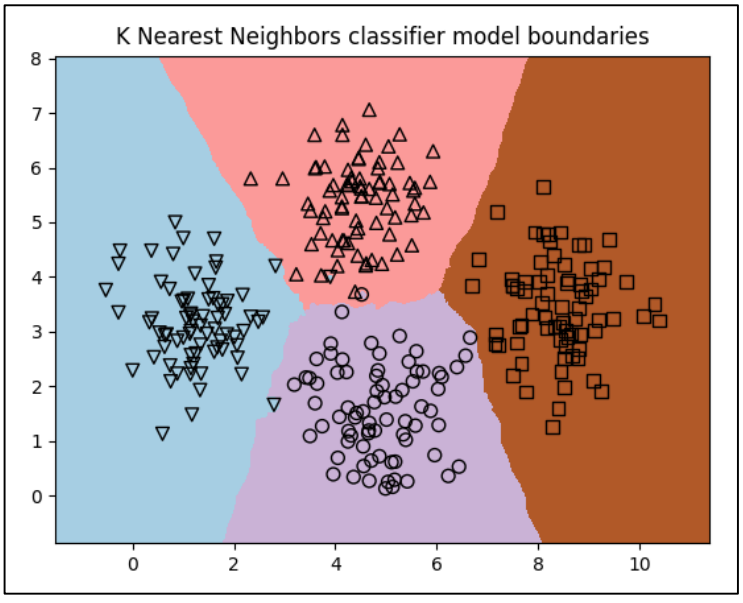


Рис. 8.2. Вхідні дані

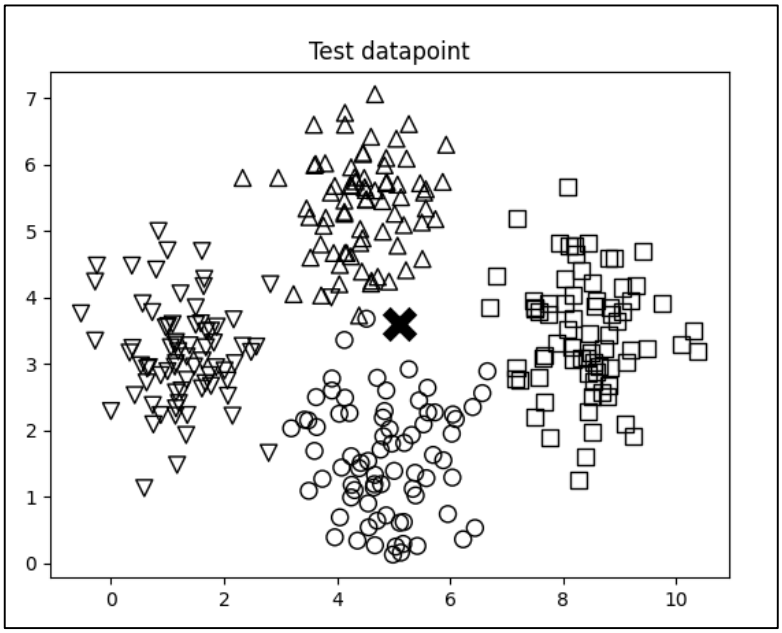


Рис. 8.3. Тестова точка даних

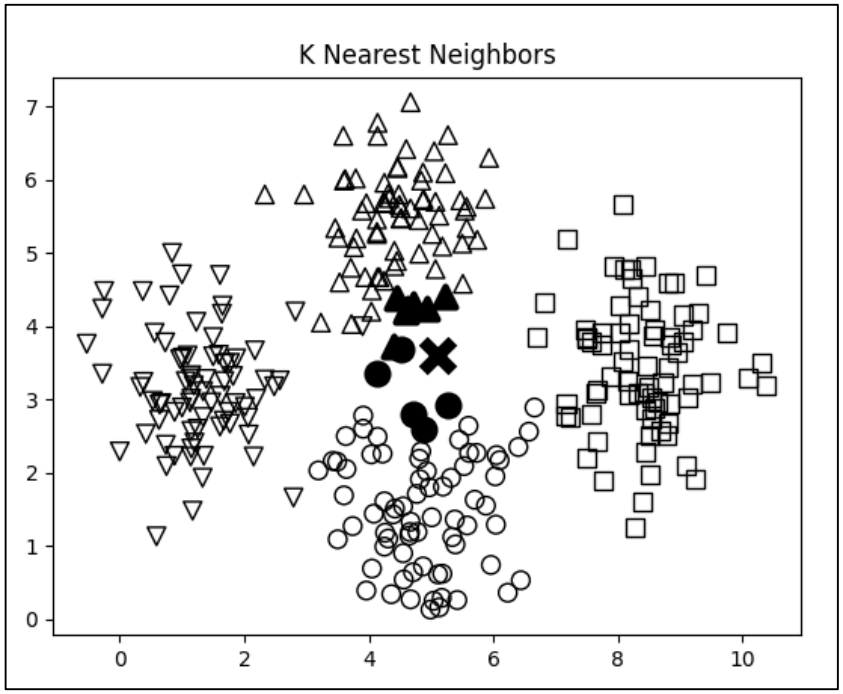


Рис. 8.4. К найближчих сусідів тестової точки

```
E:\AI-subject\L-4\tasks>python LR_4_task_8.py
Predicted output: 1

E:\AI-subject\L-4\tasks>
```

Рис. 8.5. Результат виконання програми у терміналі

Висновки щодо класифікації методом k найближчих сусідів

На першому зображенні відтворені вхідні дані, де точки з різними символами відображають різні категорії даних. Другий графік показує модель класифікатора з використанням методу k-найближчих сусідів. Ця модель визначає межі між категоріями на основі навчальних даних, розділяючи область на різні сегменти, які відповідають відмінним категоріям. Третє зображення показує тестову точку даних (X) поруч з навчальними точками. Четвертий графік демонструє k найближчих сусідів тестової точки поруч з навчальними точками. Цей графік відображає, які саме точки з навчального набору були обрані як найбільш близькі до тестової точки для прийняття рішення щодо її класифікації. Інформація у терміналі підтверджує, що тестова точка [5.1, 3.6] належить до класу "1".

Завдання 2.9. **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


# 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
рівною 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] - da-
taset[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
рівною 0
    if num_ratings == 0:
        return 0

    # Обчислення суми рейтингових оцінок усіх фільмів, оцінених обома
користувачами
    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
кори-стувачами
    user1_squared_sum = np.sum([np.square(dataset[user1][item]) for item in com-
mon_movies])
    user2_squared_sum = np.sum([np.square(dataset[user2][item]) for item in com-
mon_movies])

    # Обчислення суми творів рейтингових оцінок всіх фільмів, оцінених обома
користува-чами
    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))
```

Результат виконання програми з різними параметрами:

```
E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Euclidean

Euclidean score:
0.585786437626905

E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Pearson

Pearson score:
0.9909924304103233

E:\AI-subject\L-4\tasks>
```

Рис. 9.1. Результат виконання програми для David Smith та Bill Duffy

```
E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Euclidean

Euclidean score:
0.30383243470068705

E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Pearson

Pearson score:
0.7587869106393281

E:\AI-subject\L-4\tasks>
```

Рис. 9.2. Результат виконання програми для David Smith та Samuel Miller

```
E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Euclidean

Euclidean score:
0.2857142857142857

E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Pearson

Pearson score:
0

E:\AI-subject\L-4\tasks>
```

Рис. 9.3. Результат виконання програми для David Smith та Julie Hammel

```
E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Euclidean

Euclidean score:
0.28989794855663564

E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearson

Pearson score:
0.6944217062199275

E:\AI-subject\L-4\tasks>
```

Рис. 9.4. Результат виконання програми для David Smith та Clarissa Jackson

```
E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Euclidean

Euclidean score:
0.38742588672279304

E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Pearson

Pearson score:
0.9081082718950217

E:\AI-subject\L-4\tasks>
```

Рис. 9.5. Результат виконання програми для David Smith та Adam Cohen

```
E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Chris Duncan" --score-type Euclidean

Euclidean score:
0.38742588672279304

E:\AI-subject\L-4\tasks>python LR_4_task_9.py --user1 "David Smith" --user2 "Chris Duncan" --score-type Pearson

Pearson score:
1.0

E:\AI-subject\L-4\tasks>
```

Рис. 9.6. Результат виконання програми для David Smith та Chris Duncan

Висновки щодо обчислення ознак подібності

Отже, метод Евкліда вимірює схожість між об'єктами, оцінюючи відстань між їх атрибутами у просторі. Коефіцієнт кореляції Пірсона враховує лінійний статистичний зв'язок між ознаками, що визначає, наскільки одна змінна залежить від іншої. У нашому випадку високий коефіцієнт кореляції Пірсона свідчить про значну взаємозалежність користувачів, що підтверджує більшу подібність їх вподобань.

Завдання 2.10. **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 to
the in-put user')
    parser.add_argument('--user', dest='user', required=True,
                        help='Input user')
    return parser


# Знаходження користувачів у наборі даних, схожих на введеного користувача
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))
```

Результат виконання програми:

```
E:\AI-subject\L-4\tasks>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

E:\AI-subject\L-4\tasks>
```

Рис. 10.1. Пошук користувачів схожих на Bill Duffy

```
E:\AI-subject\L-4\tasks>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

E:\AI-subject\L-4\tasks>
```

Рис. 10.2. Пошук користувачів схожих на Clarissa Jackson

Висновки щодо результатів пошуку користувачів зі схожими уподобаннями методом колаборативної фільтрації

Результати пошуку методом колаборативної фільтрації свідчать про значну подібність у вподобаннях між парами користувачів. Наприклад, Clarissa Jackson має найбільшу схожість з Chris Duncan, що може вказувати на схожість їхніх виборів контенту. Так само, Bill Duffy найбільше схожий на David Smith, що також свідчить про високу ступінь спільних інтересів у певній області.

Завдання 2.11. 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


# 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)
```

Результат виконання програми:

```
E:\AI-subject\L-4\tasks>python LR_4_task_11.py --user "Chris Duncan"

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

Рис. 11.1. Рекомендації для Chris Duncan

```
E:\AI-subject\L-4\tasks>python LR_4_task_11.py --user "Julie Hammel"

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

E:\AI-subject\L-4\tasks>
```

Рис. 11.2. Рекомендації для Julie Hammel

Висновки щодо результатів створення рекомендацій

Результати аналізу системи рекомендацій показують, що вона успішно враховує уподобання користувачів та надає їм персоналізовані рекомендації.

Наприклад, для користувача Chris Duncan в системі пропонується перелік фільмів, які, ймовірно, зацікавлять його, враховуючи його схожість з іншими користувачами. Те ж саме відбувається з користувачем Julie Hammel. Ці результати підтверджують ефективність методів систем рекомендацій у створенні персоналізованих рекомендацій.

Висновок: у ході виконання лабораторної роботи я, використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив методи ансамблів у машинному навчанні та створив рекомендаційні системи.