ЛАБОРАТОРНА РОБОТА № 4

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:

Завдання 2.1. Створення класифікаторів на основі випадкових та гранично випадкових лісів

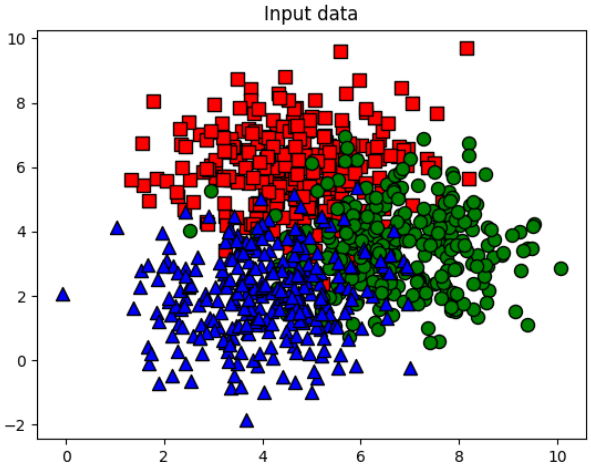


Рис. 1. Зображення розподілення даних

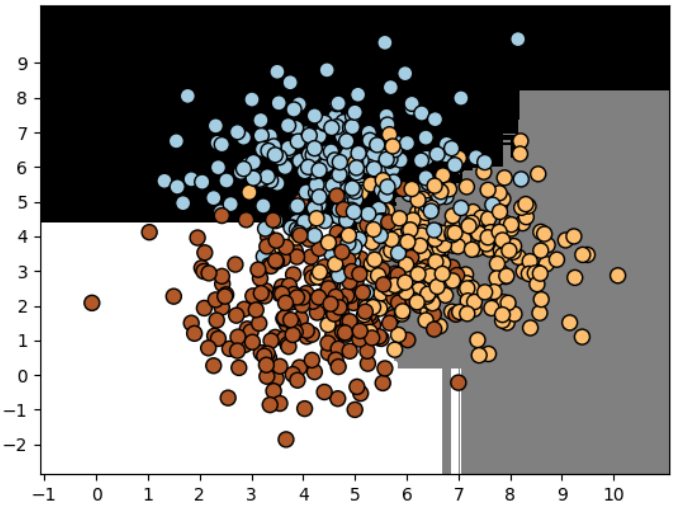


Рис. 2. Класифікація методом випадкових дерев

					ДУ «Житомирська політехніка». 23.121.8.000 – Лр4		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Палій І.В.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів
Перевір.		Голенко М.Ю.					
Керівник						ФІКТ Гр. ІПЗ-20-2	
Н. контр.							
Зав. каф.							

```
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
```

Рис. 3. Характеристики роботи методу випадкових дерев

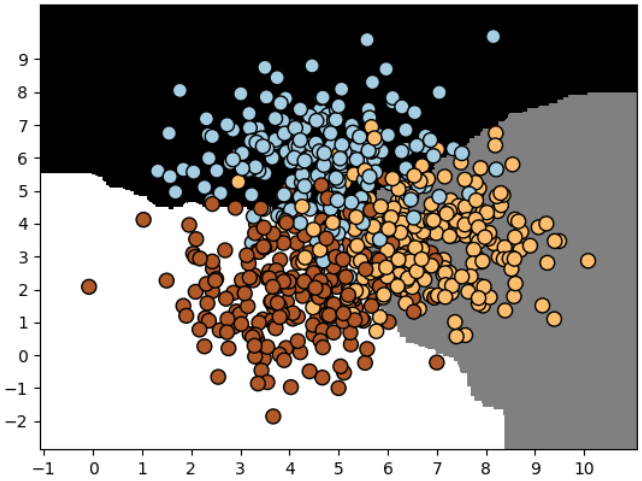


Рис. 4. Класифікація методом гранично випадкових дерев

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

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. 5. Характеристики роботи методу гранично випадкових дерев

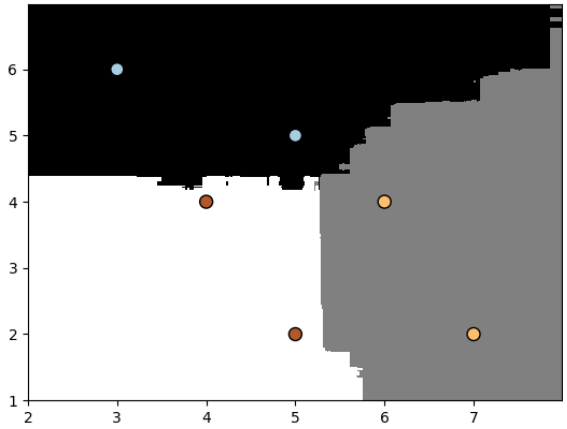


Рис. 6. Візуалізація можливих класів точок (rf)

```
Confidence measure:
Datapoint: [5 5]
Predicted class: Class-0
Probabilities: [0.81427532 0.08639273 0.09933195]
Datapoint: [3 6]
Predicted class: Class-0
Probabilities: [0.93574458 0.02465345 0.03960197]
Datapoint: [6 4]
Predicted class: Class-1
Probabilities: [0.12232404 0.7451078  0.13256816]
Datapoint: [7 2]
Predicted class: Class-1
Probabilities: [0.05415465 0.70660226 0.23924309]
Datapoint: [4 4]
Predicted class: Class-2
Probabilities: [0.20594744 0.15523491 0.63881765]
Datapoint: [5 2]
Predicted class: Class-2
Probabilities: [0.05403583 0.0931115  0.85285267]
```

Рис. 7. Дані про можливі класи (rf)

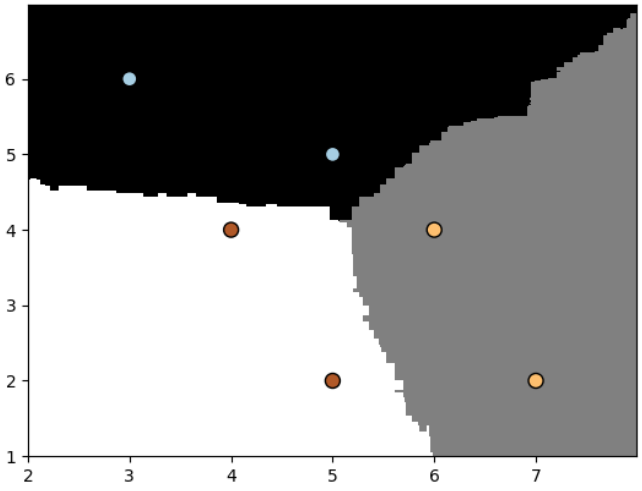


Рис. 8. Візуалізація можливих класів точок (erf)

```
Confidence measure:
Datapoint: [5 5]
Predicted class: Class-0
Probabilities: [0.48904419 0.28020114 0.23075467]
Datapoint: [3 6]
Predicted class: Class-0
Probabilities: [0.66707383 0.12424406 0.20868211]
Datapoint: [6 4]
Predicted class: Class-1
Probabilities: [0.25788769 0.49535144 0.24676087]
Datapoint: [7 2]
Predicted class: Class-1
Probabilities: [0.10794013 0.6246677  0.26739217]
Datapoint: [4 4]
Predicted class: Class-2
Probabilities: [0.33383778 0.21495182 0.45121039]
Datapoint: [5 2]
Predicted class: Class-2
Probabilities: [0.18671115 0.28760896 0.52567989]
```

Рис. 9. Дані про можливі класи (erf)

Використання випадкових дерев та граничних випадкових дерев дозволяє ефективно проводити класифікацію даних, і з цих двох методів останній виявляється більш ефективним.

Код програми:

```python
import argparse
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from utilities import visualize_classifier
from sklearn.model_selection import cross_val_score, train_test_split

# Парсер аргументів
1 usage
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Classify data using Ensemble Learning techniques')
    parser.add_argument( *name_or_flags: "--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
    # Завантаження вхідних даних
    input_file = 'data_random_forests.txt'
    data = np.loadtxt(input_file, delimiter=',')
    X, Y = data[:, :-1], data[:, -1]
    print(X)
    # 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='red', edgecolors='black', linewidth=1, marker='s')
    plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='green', edgecolors='black', linewidth=1, marker='o')
    plt.scatter(class_2[:, 0], class_2[:, 1], s=75, facecolors='blue', edgecolors='black', linewidth=1, marker='^')
plt.title('Input data')
plt.show()
# Розбивка даних на навчальний та тестовий набори
X_train, X_test, Y_train, Y_test = train_test_split( *arrays: 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)

# Перевірка роботи класифікатора
class_names = ['Class-0', 'Class-1', 'Class-2']
print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
Y_train_pred = classifier.predict(X_train)
print(classification_report(Y_train, Y_train_pred, target_names=class_names))
print("#" * 40 + "\n")

print("#" * 40)
print("\nClassifier performance on test dataset\n")
Y_test_pred = classifier.predict(X_test)
print(classification_report(Y_test, Y_test_pred, target_names=class_names))
print("#" * 40 + "\n")
```

Завдання 2.2. Обробка дисбалансу класів

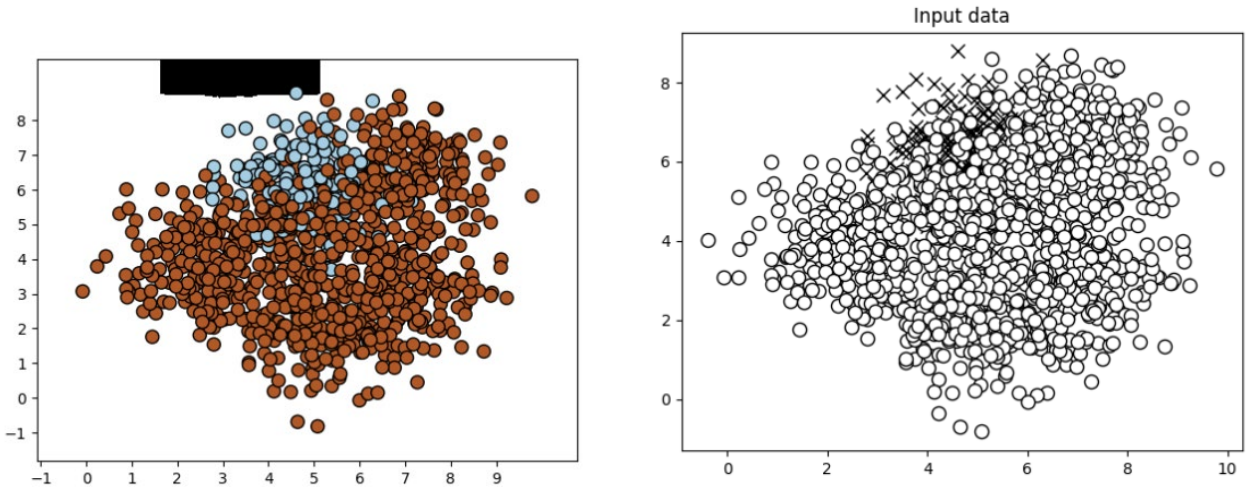


Рис. 10-11: Розподілення незбалансованих даних

```
              precision    recall  f1-score   support

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

    accuracy                           0.82       375
   macro avg       0.41      0.50      0.45       375
weighted avg       0.67      0.82      0.73       375

########################################
Classifier performance on test dataset
              precision    recall  f1-score   support

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

    accuracy                           0.82       375
   macro avg       0.41      0.50      0.45       375
weighted avg       0.67      0.82      0.73       375

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

Рис. 12. Характеристики збалансованої класифікаці

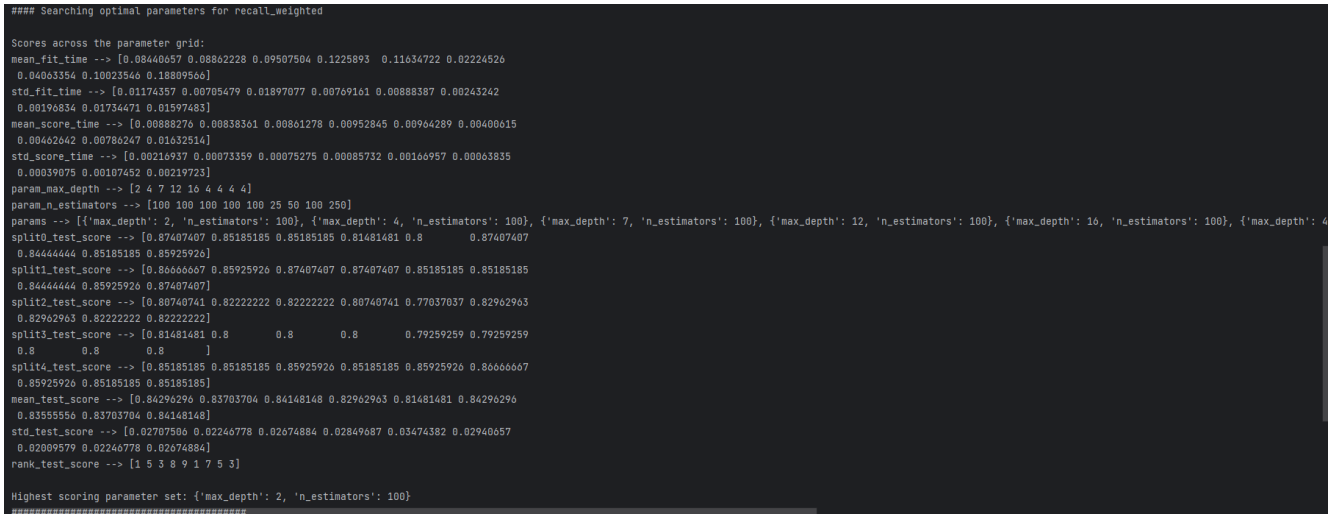
Код програми:

```
import sys
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report
from utilities import visualize_classifier

if __name__ == '__main__':
    # Завантаження вхідних даних
    input_file = 'data_imbalance.txt'
    data = np.loadtxt(input_file, delimiter=',')
    X, Y = data[:, :-1], data[:, -1]
    # Поділ вхідних даних на два класи на підставі міток
    class_0 = np.array(X[Y == 0])
    class_1 = np.array(X[Y == 1])
    # Візуалізація вхідних даних
    plt.figure()
    plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='black',
edgecolors='black', linewidth=1, marker='x')
    plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
edgecolors='black', linewidth=1, marker='o')
    plt.title('Input data')
    # Розбиття даних на навчальний та тестовий набори
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25,
random_state=5)
    # Класифікатор на основі гранично випадкових лісів
    params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}

    if len(sys.argv) > 1:
        if sys.argv[1] == 'balance':
            params['class_weight'] = 'balanced'
        else:
            raise TypeError("Invalid input argument; should be 'balance' or
nothing")

    classifier = ExtraTreesClassifier(**params)
    classifier.fit(X_train, Y_train)
    visualize_classifier(classifier, X_train, Y_train)

    Y_test_pred = classifier.predict(X_test)
    # Обчислення показників ефективності класифікатора
    class_names = ['Class-0', 'Class-1']
    print("\n" + "#"*40)
    print("Classifier performance on training dataset")
    print(classification_report(Y_test, Y_test_pred, target_names=class_names))
    print("#"*40)
    print("Classifier performance on test dataset")
    print(classification_report(Y_test, Y_test_pred, target_names=class_names))
    print("#"*40 + "\n")
    plt.show()
```

Висновок до завдання: Дані були класифіковані коректно та ефективно завдяки балансуванню.

Завдання 2.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку



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

Scores across the parameter grid:
mean_fit_time --> [0.08440657 0.08862228 0.09507504 0.1225893  0.11634722 0.02224526
 0.04063354 0.10023546 0.18809566]
std_fit_time --> [0.01174357 0.00705479 0.01897077 0.00769161 0.00888387 0.00243242
 0.00196834 0.01734471 0.01597483]
mean_score_time --> [0.00888276 0.00838361 0.00861278 0.00952845 0.00964289 0.00400615
 0.00462642 0.00786247 0.01632514]
std_score_time --> [0.00216937 0.00073359 0.00075275 0.00085732 0.00166957 0.00063835
 0.00039075 0.00107452 0.00219723]
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
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. 13. Отримання даних процесу класифікації

```
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. 14. Характеристика класифікації зі сітковим пошуком

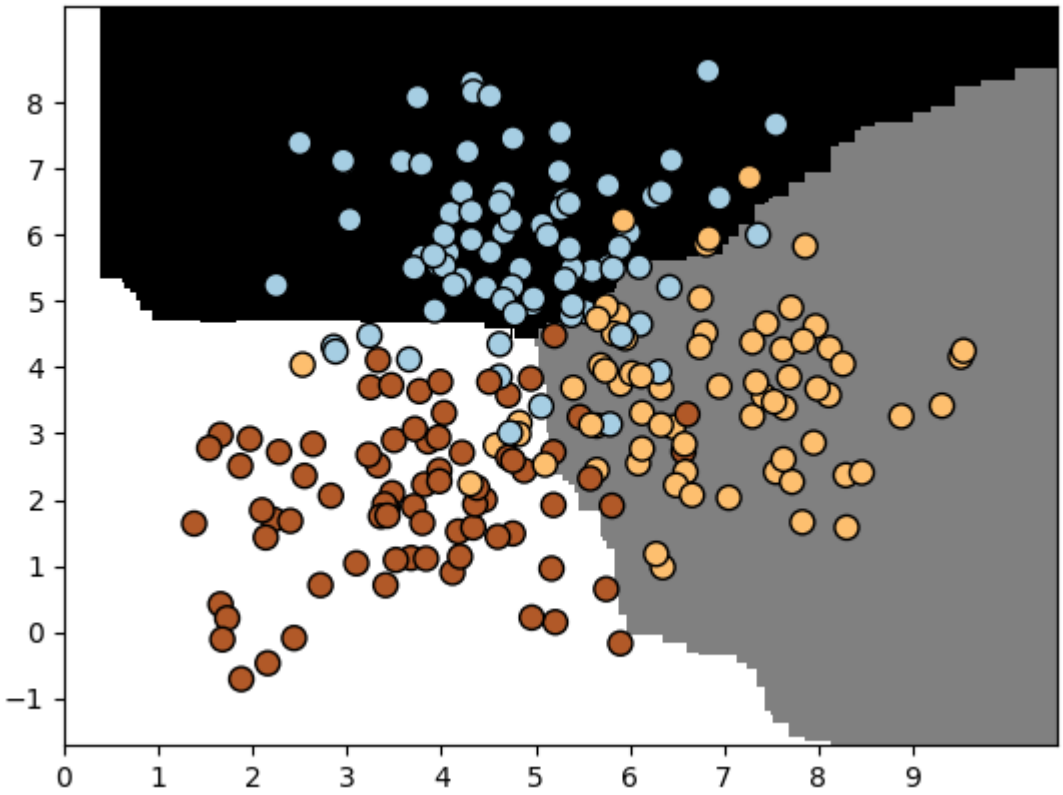


Рис. 15. Візуалізація класифікації даних зі сітковим пошуком

Код програми:

```
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report
from utilities import visualize_classifier

input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, Y = data[:, :-1], data[:, -1]
# 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_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)
```

Завдання 2.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів

```
Mean absolute error = 7.42
Predicted traffic: 26
```

Рис. 16. Результат регресії на основі гранично випадкових лісів

Код програми:

```
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesRegressor
from sklearn import preprocessing

input_file = 'traffic_data.txt'
data = []
with open(input_file, 'r') as f:
    for line in f.readlines():
        items = line[:-1].split(',')
        data.append(items)

data = np.array(data)

label_encoder = []
X_encoded = np.empty(data.shape)
for i, item in enumerate(data[0]):
    if item.isdigit():
        X_encoded[:, i] = data[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(data[:, i])
```

```
X = X_encoded[:, :-1].astype(int)
Y = X_encoded[:, -1].astype(int)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25,
random_state=5)
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, Y_train)

Y_pred = regressor.predict(X_test)
print("Mean absolute error =", round(mean_absolute_error(Y_test, Y_pred), 2))

test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
test_datapoint_encoded = [-1] * len(test_datapoint)
count = 0

for i, item in enumerate(test_datapoint):
    if item.isdigit():
        test_datapoint_encoded[i] = int(test_datapoint[i])
    else:
        test_datapoint_encoded[i] =
int(label_encoder[count].transform([test_datapoint[i]]))
        count = count + 1

test_datapoint_encoded = np.array(test_datapoint_encoded)

print("Predicted traffic:", int(regressor.predict([test_datapoint_encoded])[0]))
```

Висновок до завдання: Отримано число 26, воно є дуже близьким до фактичного значення.

Завдання 2.6. Створення навчального конвеєра (конвеєра машинного навчання)

```
Predicted output: [0 2 2 0 2 0 2 1 0 1 1 2 0 0 2 2 1 0 0 1 0 2 1 0 2 2 0 0 1 2 1 2 1 0 2 2 1
 1 2 2 2 0 1 2 2 1 1 2 1 0 1 2 2 2 2 0 2 2 0 2 2 0 1 0 2 2 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 0 2 0 2 1 2 1 0 0 1 1 1 1 2 2 2 2 0 1 1
 0 2 1 0 0 1 1 1 1 0 0 0 1 2 1 1 0 2 1 2 0 0 1 0 1 1 0 1 1 1 2 2 0 0 1 2 0
 2 2]
Score: 0.8666666666666667
Selected features: [4, 7, 8, 12, 14, 17, 22]
```

Рис. 17: Отримані результати навчального конвеєра

Код 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
```

```
# Генерування 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=10)
# 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
print("Predicted output:", processor_pipeline.predict(X))
# Виведення оцінки
print("Score:", 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("Selected features:", selected)
```

Висновок до завдання: Вибрані найбільш важливі ознаки з вхідних даних.

Завдання 2.7. Пошук найближчих сусідів

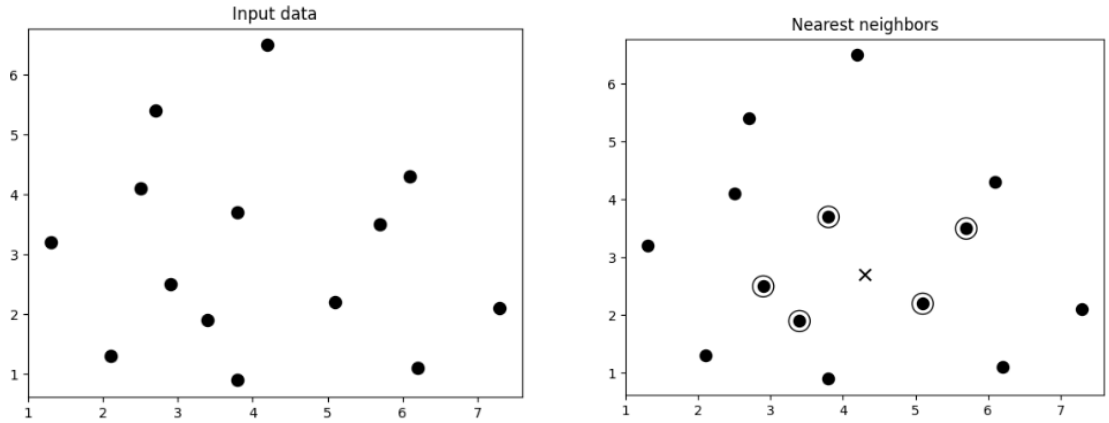


Рис. 18-19: Результат програми та графіки

```
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]
```

Рис. 20. Дані про найближчих сусідів

Код програми:

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

# Input data
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]])

# Number of nearest neighbors
k = 5

# Test datapoint
test_datapoint = [4.3, 2.7]

# Plot input data
plt.figure()
plt.title('Input data')
plt.scatter(X[:,0], X[:,1], marker='o', s=75, color='black')

# Build K Nearest Neighbors model
knn_model = NearestNeighbors(n_neighbors=k, algorithm='ball_tree').fit(X)
distances, indices = knn_model.kneighbors([test_datapoint])

# Print the 'k' nearest neighbors
print("\nK Nearest Neighbors:")
for rank, index in enumerate(indices[0][:k], start=1):
    print(str(rank) + " ==>", X[index])

# Visualize the nearest neighbors along with the test datapoint
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()
```

Висновок до завдання: Графік 18 – вхідні дані. Графік 19 - найближчі сусіди. В терміналі були вивединні координати.

Завдання 2.8. Створити класифікатор методом *k* найближчих сусідів

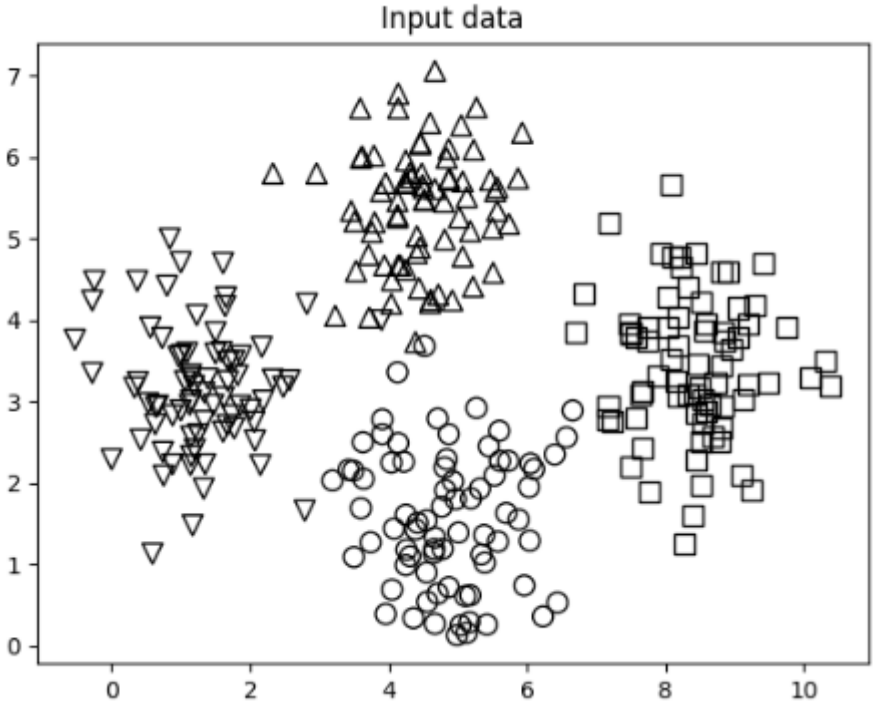


Рис. 21. Вхідні дані

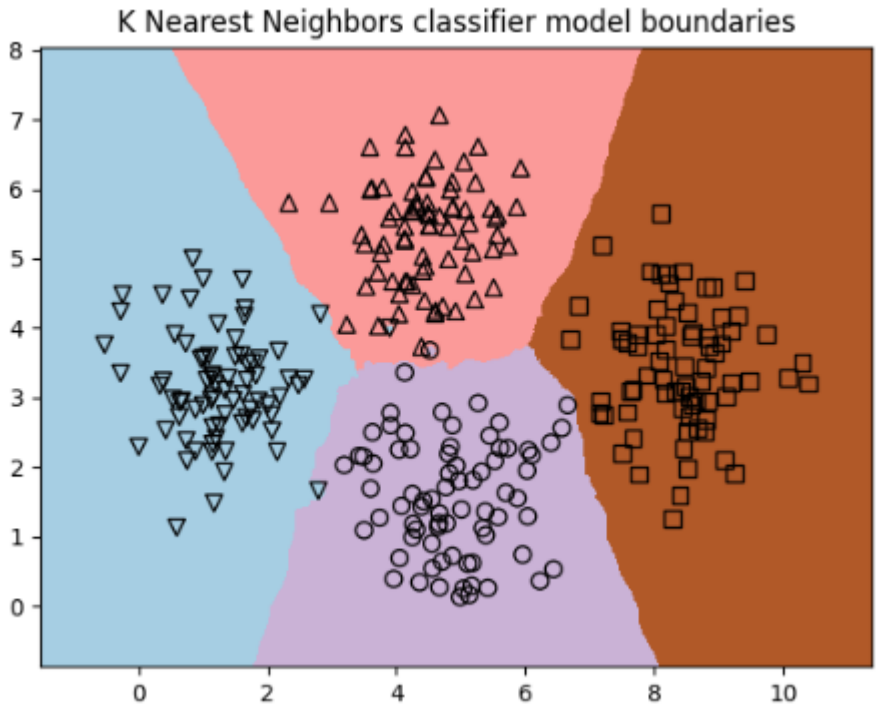


Рис. 22. Межі класів

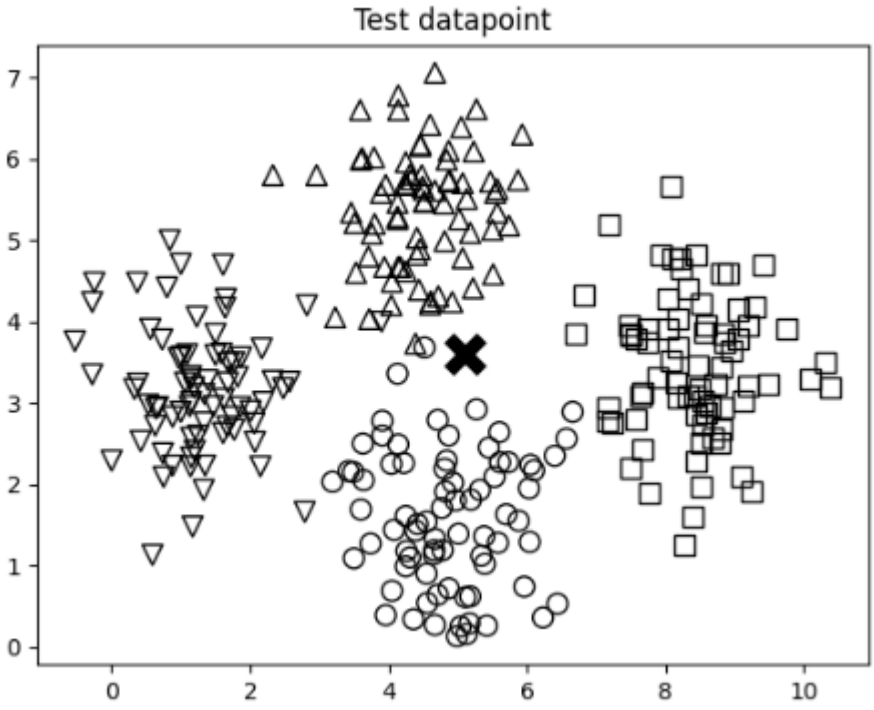


Рис. 23. Тестова точка даних

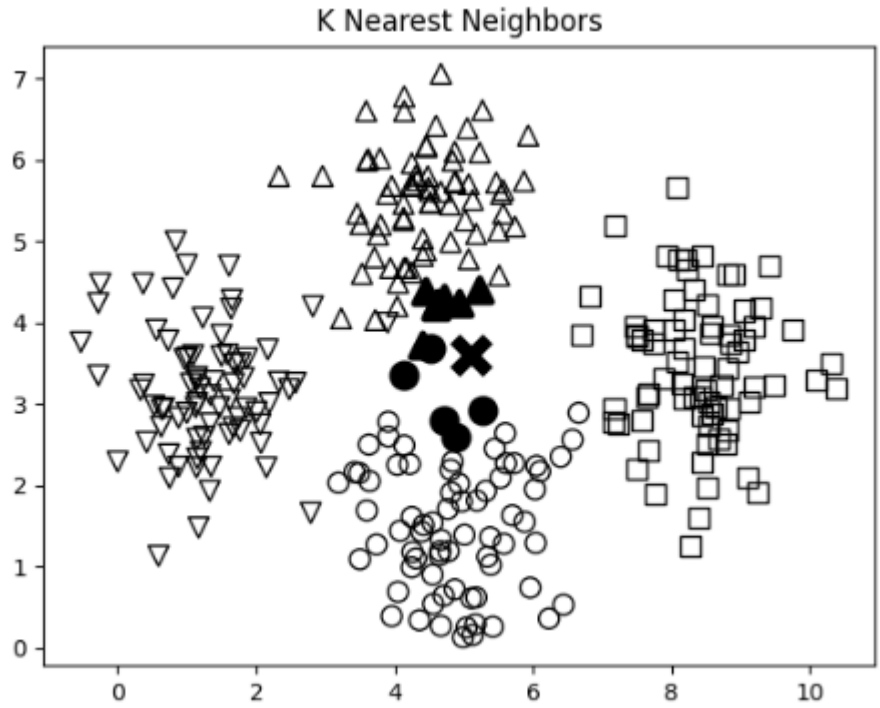


Рис. 24. Найближчі сусіди введеної точки

```
Predicted output: 1
```

Рис. 25. Обрахований клас точки

Код програми:

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

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

# Plot input data
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')

# Number of nearest neighbors
num_neighbors = 12
```

```
# Step size of the visualization grid
step_size = 0.01

# Create a K Nearest Neighbours classifier model
classifier = neighbors.KNeighborsClassifier(num_neighbors, weights='distance')

# Train the K Nearest Neighbours model
classifier.fit(X, y)

# Create the mesh to plot the boundaries
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))

# Evaluate the classifier on all the points on the grid
output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])

# Visualize the predicted output
output = output.reshape(x_values.shape)
plt.figure()
plt.pcolormesh(x_values, y_values, output, cmap=cm.Paired)

# Overlay the training points on the map
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')

# Test input datapoint
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')

# Extract the K nearest neighbors
_, indices = classifier.kneighbors([test_datapoint])
indices = indices.astype(int)[0]

# Plot k nearest neighbors
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()
```

Завдання 2.9. Обчислення оцінок подібності

```
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Euclidean

Euclidean score:
0.585786437626905
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Pearson

Pearson score:
0.9909924304103233
```

Рис. 26. Обрахунок оцінок для David Smith та Bill Duffy

```
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Brenda Peterson" --score-type Euclidean

Euclidean score:
0.1424339656566283
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Brenda Peterson" --score-type Pearson

Pearson score:
-0.7236759610155113
```

Рис. 27. Обрахунок оцінок для David Smith та Brenda Peterson

```
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Euclidean

Euclidean score:
0.30383243470068705
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Pearson

Pearson score:
0.7587869106393281
```

Рис. 28. Обрахунок оцінок для David Smith та Samuel Miller

```
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Euclidean

Euclidean score:
0.2857142857142857
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Pearson

Pearson score:
0
```

Рис. 29. Обрахунок оцінок для David Smith та Julie Hammel

```
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Euclidean

Euclidean score:
0.28989794855663564
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearson

Pearson score:
0.6944217062199275
```

Рис. 30. Обрахунок оцінок для David Smith та Clarissa Jackson

```
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Euclidean

Euclidean score:
0.38742588672279304
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Pearson

Pearson score:
0.9081082718950217
```

Рис. 31. Обрахунок оцінок для David Smith та Adam Cohen

```
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Chris Duncan" --score-type Euclidean

Euclidean score:
0.38742588672279304
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_9.py --user1 "David Smith" --user2 "Chris Duncan" --score-type Pearson

Pearson score:
1.0
```

Рис. 32. Обрахунок оцінок для David Smith та Chris Duncan

Код 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

# Compute the Euclidean distance score between user1 and 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')

    # Movies rated by both user1 and user2
    common_movies = {}

    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1

    # If there are no common movies between the users,
```

```
    # then the score is 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)))

# Compute the Pearson correlation score between user1 and user2
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')

    # Movies rated by both user1 and user2
    common_movies = {}

    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1

    num_ratings = len(common_movies)

    # If there are no common movies between user1 and user2, then the score is 0
    if num_ratings == 0:
        return 0

    # Calculate the sum of ratings of all the common movies
    user1_sum = np.sum([dataset[user1][item] for item in common_movies])
    user2_sum = np.sum([dataset[user2][item] for item in common_movies])

    # Calculate the sum of squares of ratings of all the common movies
    user1_squared_sum = np.sum([np.square(dataset[user1][item]) for item in
common_movies])
    user2_squared_sum = np.sum([np.square(dataset[user2][item]) for item in
common_movies])

    # Calculate the sum of products of the ratings of the common movies
    sum_of_products = np.sum([dataset[user1][item] * dataset[user2][item] for item
in common_movies])

    # Calculate the Pearson correlation score
    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))
```

Завдання 2.10. Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації

```
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> 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
```

Рис. 33. Знаходження користувачів схожих на Bill Duffy

```
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> 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
```

Рис. 34. Знаходження користувачів схожих на Clarissa Jackson

Код 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 input user')
    parser.add_argument('--user', dest='user', required=True,
            help='Input user')
    return parser

# Finds users in the dataset that are similar to the input user
def find_similar_users(dataset, user, num_users):
    if user not in dataset:
        raise TypeError('Cannot find ' + user + ' in the dataset')

    # Compute Pearson score between input user
    # and all the users in the dataset
    scores = np.array([[x, pearson_score(dataset, user,
            x)] for x in dataset if x != user])

    # Sort the scores in decreasing order
    scores_sorted = np.argsort(scores[:, 1])[::-1]

    # Extract the top 'num_users' scores
    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))
```

Завдання 2.11.Створення рекомендаційної системи фільмів

```
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_11.py --user "Chris Duncan"

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

Рис. 35. Рекомендації для Chris Duncan

```
PS C:\Users\ivanp\Desktop\Python A.I\Lab 4> python LR_4_task_11.py --user "Julie Hammel"

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

Рис. 36. Рекомендації для Julie Hammel

Код 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

# Get movie recommendations for the input user
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']

    # Generate movie ranks by normalization
    movie_scores = np.array([[score / similarity_scores[item], item]
                             for item, score in overall_scores.items()])

    # Sort in decreasing order
    movie_scores = movie_scores[np.argsort(movie_scores[:, 0])[::-1]]

    # Extract the movie recommendations
    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)
```

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

Посилання на репозиторій GitHub: https://github.com/IvanPaliy/A.I.-Lab-4-IPZ-Palii.git