ЛАБОРАТОРНА РОБОТА № 5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

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

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

```
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


# Функція для візуалізації меж класифікації
def plot_decision_boundaries(classifier, X, y, title, subplot_position):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax = plt.subplot(3, 2, subplot_position)
    ax.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
    scatter = ax.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap=plt.cm.Paired)
    ax.set_title(title)
    plt.colorbar(scatter, ax=ax)
```

```
# Завантаження вхідних даних
input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')

X, y = data[:, :-1], data[:, -1]

# Розіб'ємо вхідні дані на три класи
class_0 = X[y == 0]
class_1 = X[y == 1]
class_2 = X[y == 2]
# Візуалізуємо вхідні дані
plt.figure(figsize=(12, 6))
plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='white', edgecolors='black', linewidth=1, marker='s', label='Class-0')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white', edgecolors='black', linewidth=1, marker='o', label='Class-1')
plt.scatter(class_2[:, 0], class_2[:, 1], s=75, facecolors='white', edgecolors='black', linewidth=1, marker='^', label='Class-2')
plt.title('Вхідні дані')
plt.legend()
plt.show()
# Розділення 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}
# Створення двох класифікаторів: RandomForest та ExtraTrees
rf_classifier = RandomForestClassifier(**params)
erf_classifier = ExtraTreesClassifier(**params)
```

```
# Навчання класифікаторів
rf_classifier.fit(X_train, y_train)
erf_classifier.fit(X_train, y_train)

# Виведення діаграм для обох класифікаторів
plt.figure(figsize=(12, 12))

# Візуалізація для RandomForestClassifier (Training)
plot_decision_boundaries(rf_classifier, X_train, y_train, 'Random Forest (Training)', 1)

# Візуалізація для ExtraTreesClassifier (Training)
plot_decision_boundaries(erf_classifier, X_train, y_train, 'Extra Trees (Training)', 2)

# Візуалізація для RandomForestClassifier (Test)
plot_decision_boundaries(rf_classifier, X_test, y_test, 'Random Forest (Test)', 3)

# Візуалізація для ExtraTreesClassifier (Test)
plot_decision_boundaries(erf_classifier, X_test, y_test, 'Extra Trees (Test)', 4)

# Візуалізація для тестових точок (RandomForestClassifier)
test_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]])
plot_decision_boundaries(rf_classifier, test_datapoints, [0] * len(test_datapoints), 'Random Forest (Test Points)', 5)

# Візуалізація для тестових точок (ExtraTreesClassifier)
plot_decision_boundaries(erf_classifier, test_datapoints, [0] * len(test_datapoints), 'Extra Trees (Test Points)', 6)
```

```
plt.show()

# Оцінка на тестовому наборі для обох класифікаторів
y_test_pred_rf = rf_classifier.predict(X_test)
y_test_pred_erf = erf_classifier.predict(X_test)

# Виведення звіту для RandomForestClassifier
class_names = ['Class-0', 'Class-1', 'Class-2']
print("\n" + "#" * 40)
print("\nRandomForestClassifier performance on training dataset\n")
print(classification_report(y_train, rf_classifier.predict(X_train), tar-
get_names=class_names))
print("#" * 40 + "\n")

print("#" * 40)
print("\nRandomForestClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred_rf, target_names=class_names))
print("#" * 40 + "\n")

# Виведення звіту для ExtraTreesClassifier
print("\n" + "#" * 40)
print("\nExtraTreesClassifier performance on training dataset\n")
print(classification_report(y_train, erf_classifier.predict(X_train), tar-
get_names=class_names))
print("#" * 40 + "\n")

print("#" * 40)
print("\nExtraTreesClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred_erf, target_names=class_names))
print("#" * 40 + "\n")
```

```
# Обчислення параметрів довірливості для обох класифікаторів
print("\nConfidence measure for RandomForestClassifier:")
for datapoint in test_datapoints:
    probabilities = rf_classifier.predict_proba([datapoint])[0]
    predicted_class = 'Class-' + str(np.argmax(probabilities))
    print('\nDatapoint:', datapoint)
    print('Predicted class:', predicted_class)

print("\nConfidence measure for ExtraTreesClassifier:")
for datapoint in test_datapoints:
    probabilities = erf_classifier.predict_proba([datapoint])[0]
    predicted_class = 'Class-' + str(np.argmax(probabilities))
    print('\nDatapoint:', datapoint)
    print('Predicted class:', predicted_class)
```

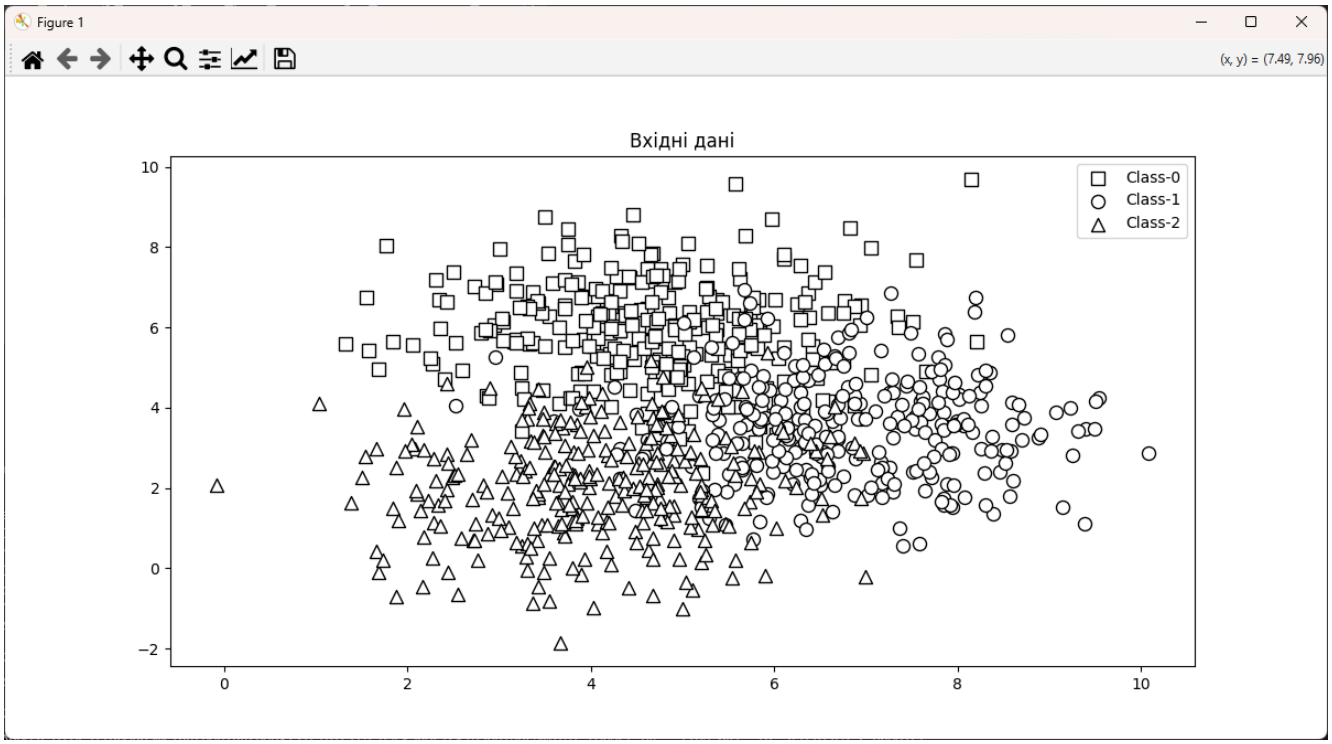

Рис. 5.1 Вхідні дані

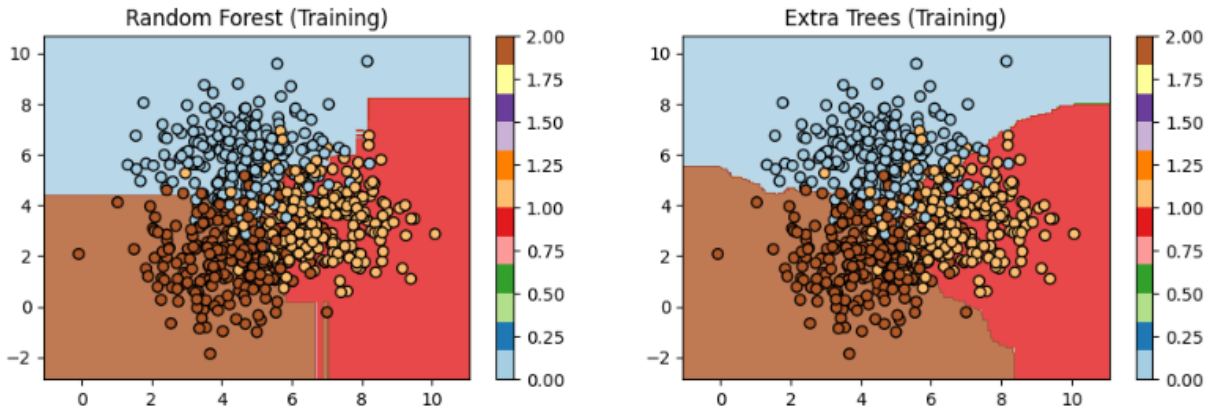

Рис. 5.2 rf – зліва, erf – зправа

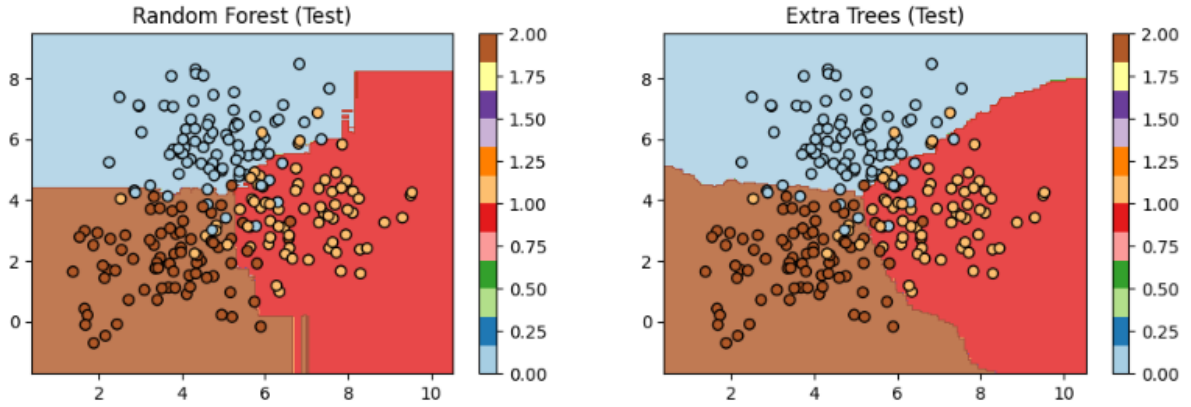

Рис. 5.3 Тест дані(rf – зліва, erf – зправа)

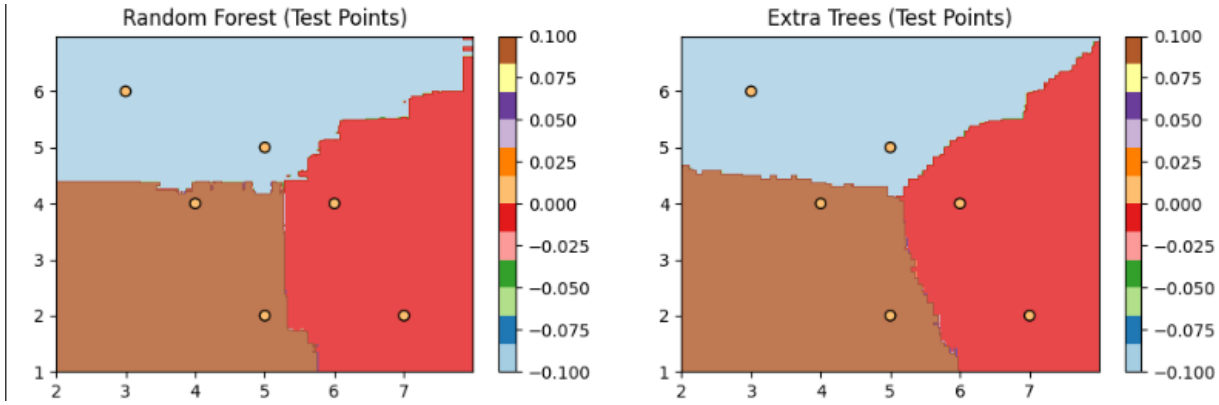

Рис. 5.4 Тест поінти (rf – зліва, erf – зправа)

***Висновок*:** Порівняння класифікаторів Random Forest (RF) та Extra Trees (ERF)

1. **Межі рішень**:
   - **Random Forest (RF):** демонструє плавні, м'які межі рішень, що свідчить про його здатність рівномірно відокремлювати класи. Це робить його менш схильним до перенавчання.

- **Extra Trees (ERF):** створює різкіші межі, які дозволяють краще захоплювати локальні особливості структури даних, але це може призводити до перенавчання на певних наборах даних.

2. **Впевненість класифікації**:
   - **RF:** забезпечує стабільні та надійні прогнози, що робить його зручним для застосування в реальних задачах.
   - **ERF:** має здатність до високої впевненості в окремих зонах, але його прогнози можуть бути менш стабільними в інших областях.
3. **Продуктивність на тестовому наборі**:
   - **RF:** демонструє високу здатність до узагальнення, що дозволяє моделі добре працювати на нових, невідомих даних.
   - **ERF:** може досягати кращих результатів на тренувальних даних, але його схильність до перенавчання може знизити продуктивність на тестовому наборі.
4. **Візуалізація результатів**:
   - **RF:** показує плавніші та передбачувані межі класифікації.
   - **ERF:** створює більш виразні та різкі межі, що ілюструє його здатність до детального захоплення структури даних.

Загальний висновок: Обидва класифікатори мають свої сильні сторони: RF — це вибір для задач, де потрібні стабільність і узагальнення, тоді як ERF підходить для задач, де важливо глибше проаналізувати структуру даних. Остаточний вибір класифікатора залежить від конкретних вимог задачі та властивостей набору даних.

Завдання 2: Створення класифікаторів на основі випадкових та гранично випадкових лісів.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils import resample

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

# Візуалізація початкових даних
class_0 = X[y == 0]
class_1 = X[y == 1]

plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75, label="Class 0", edgecolors="black")
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, label="Class 1", edgecolors="black")
plt.title("Початкові дані")
plt.legend()
plt.show()

# Аналіз дисбалансу
unique, counts = np.unique(y, return_counts=True)
print("Розподіл класів до обробки:", dict(zip(unique, counts)))

# Балансування даних
X_class_0 = X[y == 0]
X_class_1 = X[y == 1]

if counts[0] > counts[1]:
```

```
        X_class_1_resampled = resample(X_class_1, replace=True, n_samples=counts[0], random_state=42)
        X_resampled = np.vstack((X_class_0, X_class_1_resampled))
                    y_resampled = np.hstack((np.zeros(len(X_class_0)), np.ones(len(X_class_1_resampled))))
    else:
        X_class_0_resampled = resample(X_class_0, replace=True, n_samples=counts[1], random_state=42)
        X_resampled = np.vstack((X_class_0_resampled, X_class_1))
                y_resampled = np.hstack((np.zeros(len(X_class_0_resampled)), np.ones(len(X_class_1))))

    # Перевірка після балансування
    unique, counts = np.unique(y_resampled, return_counts=True)
    print("Розподіл класів після балансування:", dict(zip(unique, counts)))

    # Розділення на навчальний і тестовий набори
    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.25, random_state=42)

    # Побудова класифікатора
    classifier = RandomForestClassifier(n_estimators=100, class_weight='balanced', random_state=42)
    classifier.fit(X_train, y_train)

    # Оцінка якості моделі
    y_pred = classifier.predict(X_test)

    print("\nМатриця плутанини:")
```

```
print(confusion_matrix(y_test, y_pred))
print("\nЗвіт про класифікацію:")
print(classification_report(y_test, y_pred))

# Візуалізація меж класифікації
def plot_decision_boundaries(classifier, X, y, title):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap=plt.cm.Paired)
    plt.title(title)
    plt.show()

plot_decision_boundaries(classifier, X_test, y_test, "Межі класифікації (Тестовий набір)")
```

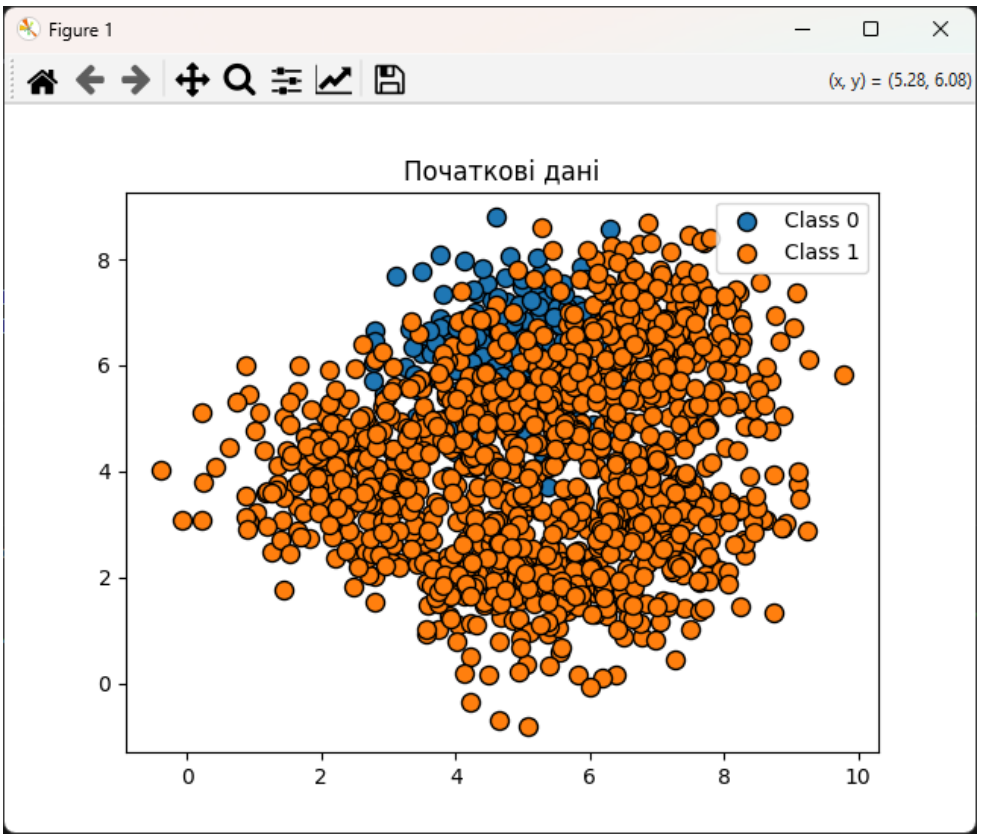

Рис. 5.5 Початкові дані

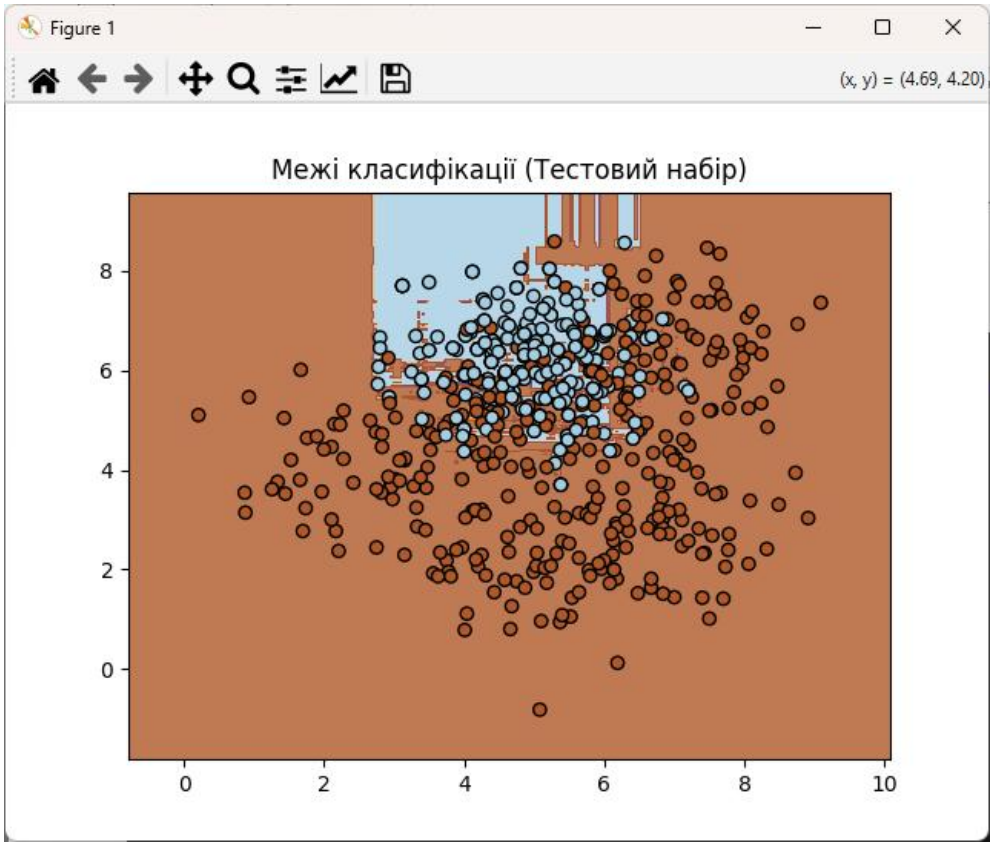

Рис. 5.6 Межі класифікації

```
PS C:\Users\TAMOTO\Desktop\University\Искуственний интеллект\lab 5>  & 'c:\Users\T
ebugpy\adapter/../..\debugpy\launcher' '57936' '--' 'c:\Users\TAMOTO\Desktop\Unive
Розподіл класів до обробки: {0.0: 250, 1.0: 1250}
Розподіл класів після балансування: {0.0: 1250, 1.0: 1250}

Матриця плутанини:
[[296   1]
 [ 37 291]]

Звіт про класифікацію:
              precision    recall  f1-score   support

         0.0       0.89      1.00      0.94       297
         1.0       1.00      0.89      0.94       328

    accuracy                           0.94       625
   macro avg       0.94      0.94      0.94       625
weighted avg       0.95      0.94      0.94       625
```

Рис. 5.7 Отриманні дані

Висновок: Модель показала високу точність та збалансованість після коригування.

Завдання 3: Обробка дисбалансу класі.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import ExtraTreesClassifier


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


# Розділення даних на навчальні та тестові 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': [25, 50, 100, 250],
    'max_depth': [2, 4, 8, 12, 16]
}

metrics = ['precision_weighted', 'recall_weighted']

# Перебір параметрів
for metric in metrics:
    print("\n### Searching optimal parameters for", metric)

    classifier = GridSearchCV(
        ExtraTreesClassifier(random_state=0),
        parameter_grid,
        cv=5,  # Кількість фолдів для крос-валідації
        scoring=metric
    )

    # Навчання моделі
    classifier.fit(X_train, y_train)

    # Виведення результатів
    print("\nGrid scores for the parameter grid:\n")
```

```
    results = classifier.cv_results_
    for mean, params in zip(results['mean_test_score'], results['params']):
        print(params, '-->', round(mean, 3))

    print("\nBest parameters for", metric, ":\n", classifier.best_params_)

# Виведення результатів роботи класифікатора
print("\nPerformance report on test set:\n")
y_pred = classifier.predict(X_test)
print(classification_report(y_test, y_pred))
```

Отримані дані:

```
### Searching optimal parameters for precision_weighted

Grid scores for the parameter grid:

{'max_depth': 2, 'n_estimators': 25} --> 0.838
{'max_depth': 2, 'n_estimators': 50} --> 0.845
{'max_depth': 2, 'n_estimators': 100} --> 0.85
{'max_depth': 2, 'n_estimators': 250} --> 0.846
{'max_depth': 4, 'n_estimators': 25} --> 0.846
{'max_depth': 4, 'n_estimators': 50} --> 0.84
{'max_depth': 4, 'n_estimators': 100} --> 0.841
{'max_depth': 4, 'n_estimators': 250} --> 0.845
{'max_depth': 8, 'n_estimators': 25} --> 0.846
{'max_depth': 8, 'n_estimators': 50} --> 0.845
{'max_depth': 8, 'n_estimators': 100} --> 0.842
{'max_depth': 8, 'n_estimators': 250} --> 0.836
```

{'max_depth': 12, 'n_estimators': 25} --> 0.83

{'max_depth': 12, 'n_estimators': 50} --> 0.827

{'max_depth': 12, 'n_estimators': 100} --> 0.832

{'max_depth': 12, 'n_estimators': 250} --> 0.828

{'max_depth': 16, 'n_estimators': 25} --> 0.811

{'max_depth': 16, 'n_estimators': 50} --> 0.818

{'max_depth': 16, 'n_estimators': 100} --> 0.816

{'max_depth': 16, 'n_estimators': 250} --> 0.817

Best parameters for precision_weighted :

{'max_depth': 2, 'n_estimators': 100}

Searching optimal parameters for recall_weighted

Grid scores for the parameter grid:

{'max_depth': 2, 'n_estimators': 25} --> 0.833

{'max_depth': 2, 'n_estimators': 50} --> 0.837

{'max_depth': 2, 'n_estimators': 100} --> 0.843

{'max_depth': 2, 'n_estimators': 250} --> 0.841

{'max_depth': 4, 'n_estimators': 25} --> 0.843

{'max_depth': 4, 'n_estimators': 50} --> 0.836

{'max_depth': 4, 'n_estimators': 100} --> 0.837

{'max_depth': 4, 'n_estimators': 250} --> 0.841

{'max_depth': 8, 'n_estimators': 25} --> 0.841

{'max_depth': 8, 'n_estimators': 50} --> 0.84

{'max_depth': 8, 'n_estimators': 100} --> 0.839

{'max_depth': 8, 'n_estimators': 250} --> 0.834

{'max_depth': 12, 'n_estimators': 25} --> 0.828

{'max_depth': 12, 'n_estimators': 50} --> 0.825

{'max_depth': 12, 'n_estimators': 100} --> 0.83

{'max_depth': 12, 'n_estimators': 250} --> 0.827

{'max_depth': 16, 'n_estimators': 25} --> 0.809

{'max_depth': 16, 'n_estimators': 50} --> 0.816

{'max_depth': 16, 'n_estimators': 100} --> 0.815

{'max_depth': 16, 'n_estimators': 250} --> 0.815

Best parameters for recall_weighted :

{'max_depth': 2, 'n_estimators': 100}

Performance report on test set:

	precision	recall	f1-score	support
0.0	0.94	0.81	0.87	79
1.0	0.81	0.86	0.83	70
2.0	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

Висновок:

Модель демонструє збалансовані результати за метриками precision і recall, що свідчить про її здатність ефективно класифікувати дані з високою точністю та чутливістю.

Завдання 4: Обчислення відносної важливості ознак.

```
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.datasets import fetch_california_housing
from sklearn.utils import shuffle

# Завантаження даних з Каліфорнійським житлом
housing = fetch_california_housing()

# Перемішування даних
X, y = shuffle(housing.data, housing.target, random_state=7)

# Розбиваємо дані 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)

# Оцінка ефективності регресора
y_pred = regressor.predict(X_test)
```

```
mse = mean_squared_error(y_test, y_pred)
evs = explained_variance_score(y_test, y_pred)

print("АДАБОСТ РЕГРЕСОР")
print("Середньоквадратична помилка:", round(mse, 2))
print("Пояснена дисперсія:", round(evs, 2))

# Отримуємо важливість ознак
feature_importances = regressor.feature_importances_
feature_names = housing.feature_names

# Нормалізуємо важливість ознак
feature_importances = 100.0 * (feature_importances / max(feature_importances))

# Сортуємо ознаки за важливістю
index_sorted = np.argsort(feature_importances)[::-1]  # Sort in descending order

# Розміщення міток уздовж осі X
pos = np.arange(index_sorted.shape[0]) + 0.5

# Побудова стовпчастої діаграми
plt.figure()
plt.barh(pos, feature_importances[index_sorted], align='center')
plt.yticks(pos, np.array(feature_names)[index_sorted])  # Use feature names correctly
plt.xlabel('Важливість ознак')
plt.title('Важливість ознак для моделі AdaBoost')
plt.show()
```

Отримані дані:

АДАБОСТ РЕГРЕСОР

Середньоквадратична помилка: 1.18

Пояснена дисперсія: 0.47

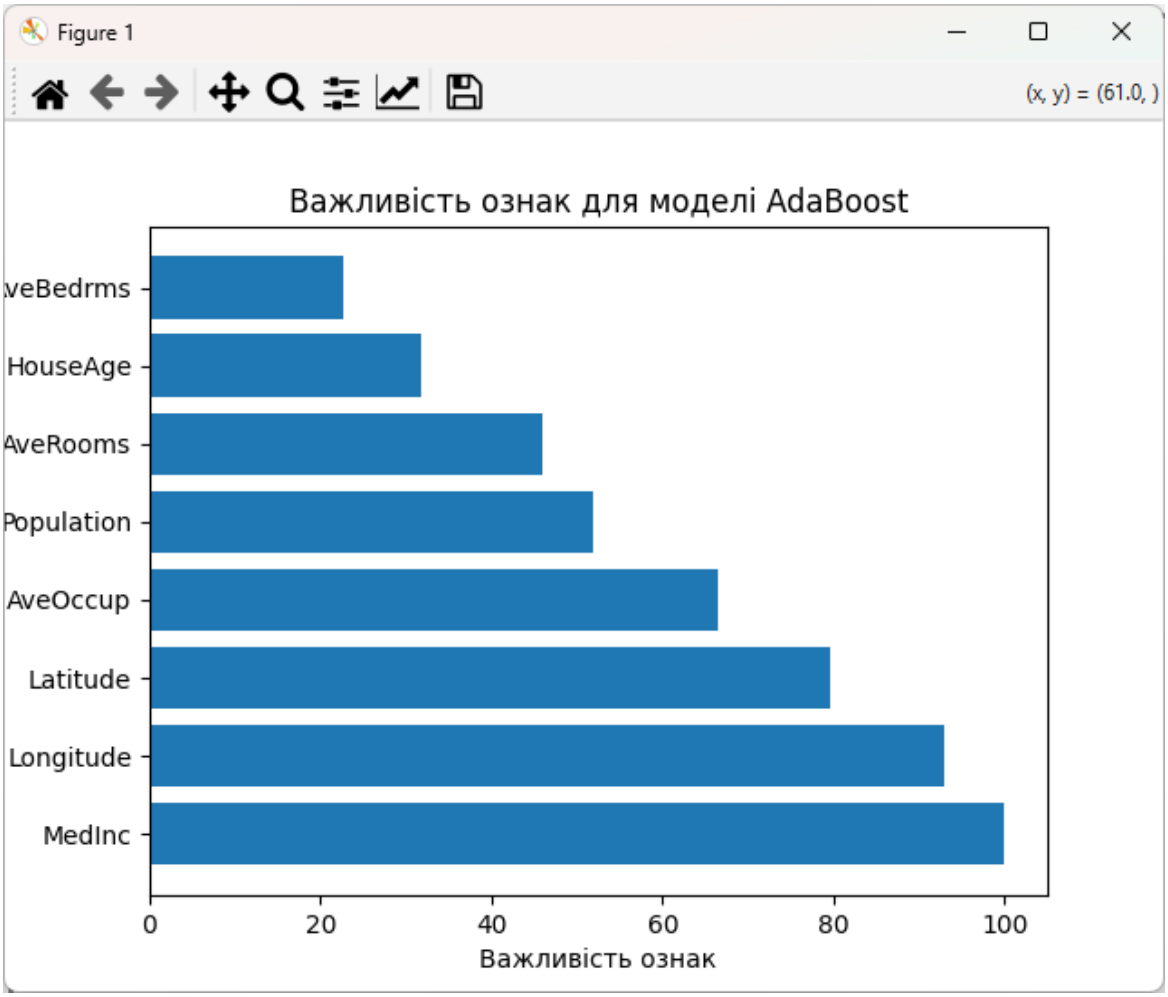

Рис. 5.8 Графік важливості ознак

Висновок:

Аналіз важливості ознак для моделі AdaBoost:

- **Найважливіші ознаки:**
  - **MedInc (медіанний дохід):** найбільший вплив на модель.
  - **Longitude (довгота)** та **Latitude (широта):** значний географічний вплив.
- **Маловажливі ознаки:**
  - **AveBedrms (середня кількість спалень):** найменший вплив.
  - **HouseAge (вік будинку):** низька значущість.

Ознаки з високою важливістю (>80) слід зберегти, а малозначущі (<20) можна виключити для спрощення моделі.

Завдання 5: Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів.

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


# Завантажимо дані із файлу traffic_data.txt
input_file = 'traffic_data.txt'
data = []
with open(input_file, 'r') as f:
    for line in f.readlines():
        items = line.strip().split(',')  # strip removes any trailing newline characters
        data.append(items)
data = np.array(data)


# Нечислові ознаки потребують кодування
label_encoder = []
X_encoded = np.empty(data.shape, dtype=object)  # Use dtype=object to hold mixed types
for i, item in enumerate(data[0]):  # Assuming the first row contains column names
    if item.isdigit():  # If the feature is numeric
        X_encoded[:, i] = data[:, i]
    else:  # If the feature is categorical
```

```
            label_encoder.append(preprocessing.LabelEncoder())
            X_encoded[:, i] = label_encoder[-1].fit_transform(data[:, i])

    # Розділення на ознаки та мітки
    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=0)

    # 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), 3))

    # Тестування кодування на одному прикладі
    test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
    test_datapoint_encoded = [-1] * len(test_datapoint)  # Initializing the encoded test datapoint
    count = 0

    for i, item in enumerate(test_datapoint):
        if not item.isdigit():  # If the feature is not numeric, it needs encoding
```

```
            test_datapoint_encoded[i] = label_encoder[count].transform([item])[0]
            count += 1
        else:
            test_datapoint_encoded[i] = int(item)

    test_datapoint_encoded = np.array(test_datapoint_encoded)

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

Отримані дані:

Mean absolute error: 7.568

Predicted traffic: 25

Висновок:

Під час лабораторної роботи, використовуючи мову програмування Python та спеціалізовані бібліотеки, було досліджено методи ансамблевого навчання в машинному навчанні.

Github - https://github.com/TAMOTO24/-Intelligen-Systems