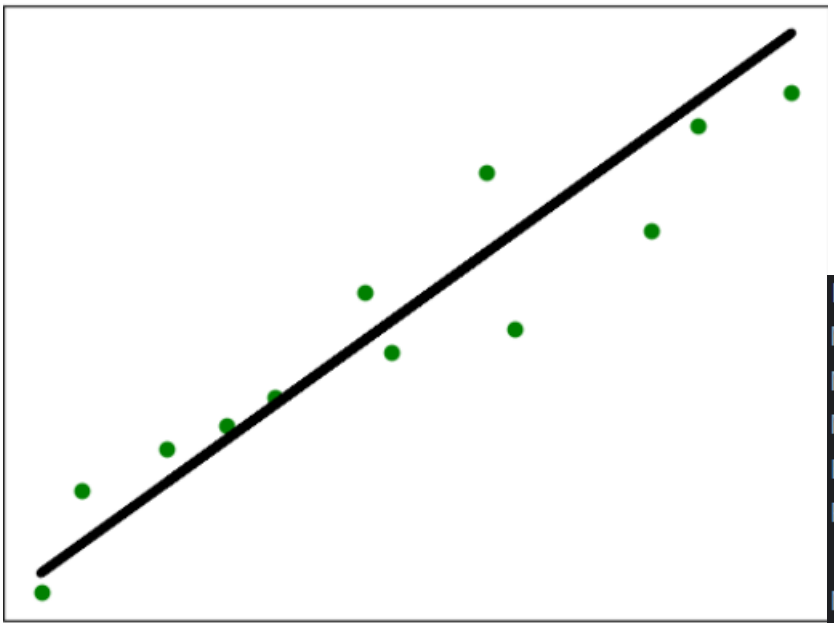
ЛАБОРАТОРНА РОБОТА № 3

ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ ТА НЕКОНТРОЬОВАНОГО НАВЧАННЯ

Мета роботи: *використовуючи спеціалізовані бібліотеки і мову програмування Python дослідити методи регресії та неконтрольованої класифікації даних у машинному навчанні.*

Хід роботи:

Завдання 2.1. Створення регресора однієї змінної



```
Linear regressor performance:
Mean absolute error = 0.59
Mean squared error = 0.49
Median absolute error = 0.51
Explain variance score = 0.86
R2 score = 0.86

New mean absolute error = 0.59
```

Рис. 1-2: Результат програми та графік

Код програми:

LR_3_task_1.py

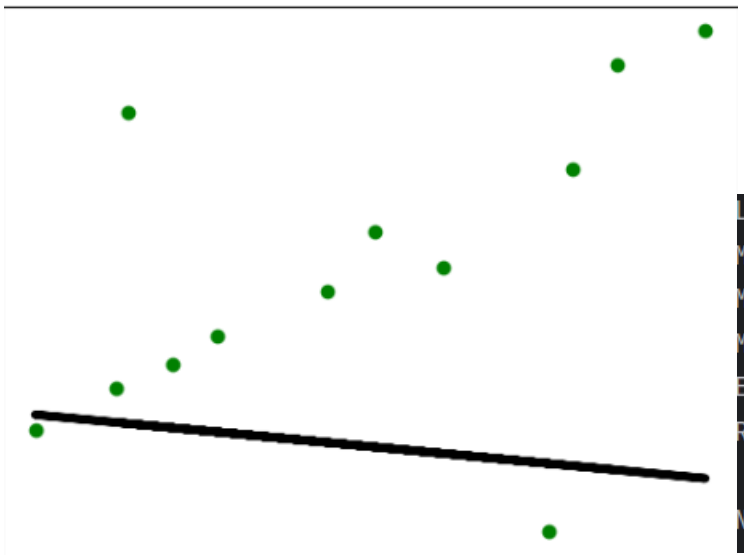
```python
import pickle
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
import matplotlib.pyplot as plt

# Вхідний файл, який містить дані
input_file = 'data_singlevar_regr.txt'

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

# Розбивка даних на навчальний та тестовий набори
num_training = int(0.8 * len(X))
num_test = len(X) - num_training

# Тренувальні дані
X_train, y_train = X[:num_training], y[:num_training]
# Тестові дані
X_test, y_test = X[num_training:], y[num_training:]

# Створення об'єкта лінійного регресора
regressor = linear_model.LinearRegression()
regressor.fit(X_train, y_train)

# Прогнозування результату
y_test_pred = regressor.predict(X_test)
```

LR_3_task_1.py

```python
y_test_pred = regressor.predict(X_test)

# Побудова графіка
plt.scatter(X_test, y_test, color='green')
plt.plot( *args: X_test, y_test_pred, color='black', linewidth=4)
plt.xticks(())
plt.yticks(())
plt.show()

print("Linear regressor performance:")
print("Mean absolute error =", round(sm.mean_absolute_error(y_test, y_test_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_test_pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, y_test_pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, y_test_pred), 2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))

# Файл для збереження моделі
output_model_file = 'model.pkl'

# Збереження моделі
with open(output_model_file, 'wb') as f: pickle.dump(regressor, f)

# Завантаження моделі
y_test_pred_new = regressor.predict(X_test)
print("\nNew mean absolute error =", round(sm.mean_absolute_error(y_test, y_test_pred_new), 2))
```

Висновок до завдання: Цей код демонструє процес навчання та використання лінійної регресії для прогнозування на основі вхідних даних, оцінки якості моделі та можливості зберігання та завантаження моделі для подальшого використання.

Завдання 2.2. Передбачення за допомогою регресії однієї змінної



```
Linear regressor performance:
Mean absolute error = 2.42
Mean squared error = 9.02
Median absolute error = 2.14
Explain variance score = -0.15
R2 score = -1.61

New mean absolute error = 2.42
```

Рис. 3-4: Результат програми та графік

Код програми:

LR_3_task_1.py LR_3_task_2.py

```python
import pickle
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
import matplotlib.pyplot as plt

# Вхідний файл, який містить дані
input_file = 'data_regr_2.txt'

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

# Розбивка даних на навчальний та тестовий набори
num_training = int(0.8 * len(X))
num_test = len(X) - num_training

# Тренувальні дані
X_train, y_train = X[:num_training], y[:num_training]

# Тестові дані
X_test, y_test = X[num_training:], y[num_training:]

# Створення об'єкта лінійного регресора
regressor = linear_model.LinearRegression()

# Тренування моделі
regressor.fit(X_train, y_train)
```

LR_3_task_1.py LR_3_task_2.py

```python
y_test_pred = regressor.predict(X_test)

# Побудова графіка
plt.scatter(X_test, y_test, color='green')
plt.plot( *args: X_test, y_test_pred, color='black', linewidth=4)
plt.xticks(())
plt.yticks(())
plt.show()

# Обрахування метрик
print("Linear regressor performance:")
print("Mean absolute error =", round(sm.mean_absolute_error(y_test, y_test_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_test_pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, y_test_pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, y_test_pred), 2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))

# Файл для збереження моделі
output_model_file = 'model.pkl'

# Збереження моделі
with open(output_model_file, 'wb') as f:
    pickle.dump(regressor, f)
# Завантаження моделі
with open(output_model_file, 'rb') as f:
    regressor_model = pickle.load(f)
# Perform prediction on test data
y_test_pred_new = regressor_model.predict(X_test)
print("\nNew mean absolute error =", round(sm.mean_absolute_error(y_test, y_test_pred_new), 2))
```

Висновок до завдання: Цей код демонструє процес навчання та використання лінійної регресії для прогнозування на основі вхідних даних, оцінки якості моделі та можливості зберігання та завантаження моделі для подальшого використання.

Завдання 2.3. Створення багатовимірного регресора

```
Linear Regressor performance:
Mean absolute error = 3.58
Mean squared error = 20.31
Median absolute error = 2.99
Explained variance score = 0.86
R2 score = 0.86

Linear regression:
 [36.05286276]

Polynomial regression:
 [41.45976677]
```

Рис. 5: Результат програми та графік

Код програми:

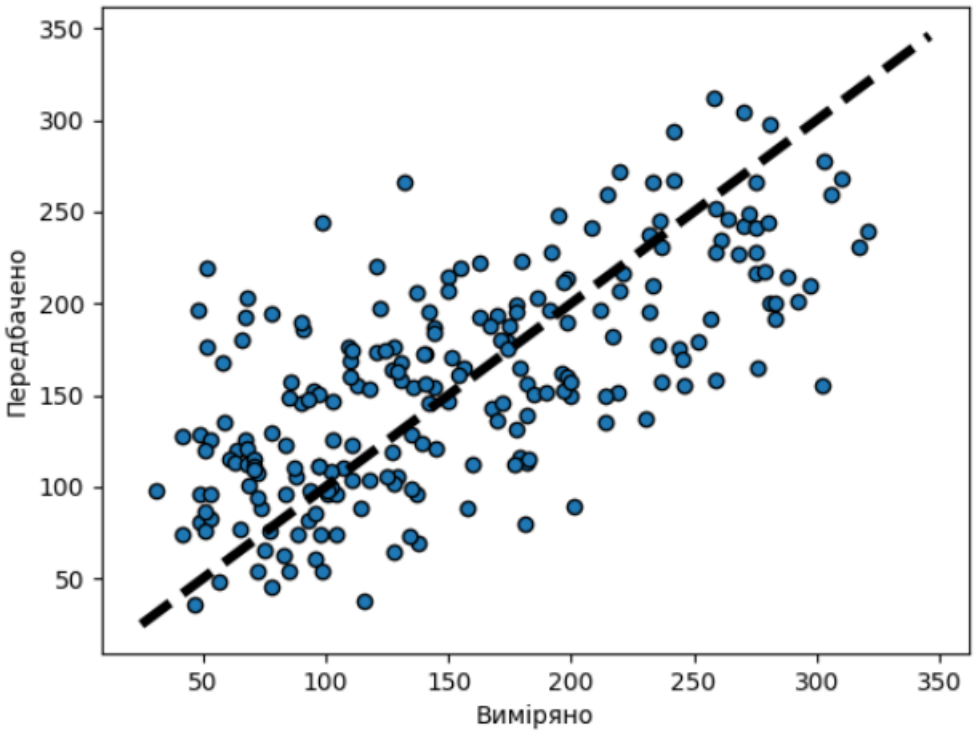
```python
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
from sklearn.preprocessing import PolynomialFeatures

# Вхідний файл, який містить дані
input_file = 'data_multivar_regr.txt'

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

# Розбивка даних на навчальний та тестовий набори
num_training = int(0.8 * len(X))
num_test = len(X) - num_training

# Тренувальні дані
X_train, y_train = X[:num_training], y[:num_training]

# Тестові дані
X_test, y_test = X[num_training:], y[num_training:]

# Створення об'єкта лінійного регресора
linear_regressor = linear_model.LinearRegression()

# Тренування моделі
linear_regressor.fit(X_train, y_train)
```

LR_3_task_1.py | LR_3_task_2.py | LR_3_task_3.py

```python
# Тренування моделі
linear_regressor.fit(X_train, y_train)

# Прогнозування результату
y_test_pred = linear_regressor.predict(X_test)

# Обрахування метрик
print("Linear Regressor performance:")
print("Mean absolute error =", round(sm.mean_absolute_error(y_test, y_test_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_test_pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, y_test_pred), 2))
print("Explained variance score =", round(sm.explained_variance_score(y_test, y_test_pred), 2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))

# Поліноміальна регресія
polynomial = PolynomialFeatures(degree=10)
X_train_transformed = polynomial.fit_transform(X_train)
datapoint = [[7.75, 6.35, 5.56]]
poly_datapoint = polynomial.fit_transform(datapoint)

poly_linear_model = linear_model.LinearRegression()
poly_linear_model.fit(X_train_transformed, y_train)
print("\nLinear regression:\n", linear_regressor.predict(datapoint))
print("\nPolynomial regression:\n", poly_linear_model.predict(poly_datapoint))
```

Висновок до завдання: У цьому коді спочатку навчається лінійна регресія на багатовимірних даних і оцінюється її продуктивність. Потім застосовується поліноміальна регресія зі степенем 10 для тих самих даних і порівнюється з лінійною регресією. Результати виводяться для обох моделей.

Завдання 2.4. Регресія багатьох змінних




```
regr.coef = [ -20.4  -265.89  564.65  325.56 -692.16  395.56   23.5   116.36  843.95
   12.72]
regr.intercept = 154.36
R2 score = 0.44
Mean absolute error = 44.8
Mean squared error = 3075.33
```

Рис. 6-7: Результат програми та графік

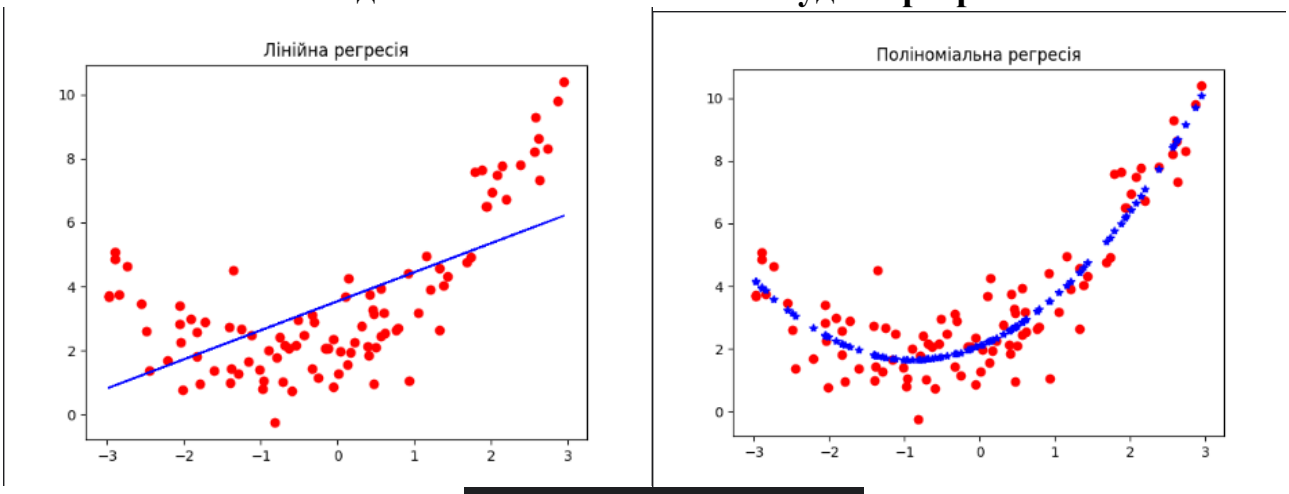
Код програми:

LR_3_task_1.py LR_3_task_2.py LR_3_task_3.py LR_3_task_4.py

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target
Xtrain, Xtest, ytrain, ytest = train_test_split( *arrays: X, y, test_size=0.5, random_state=0)
regr = linear_model.LinearRegression()
regr.fit(Xtrain, ytrain)
ypred = regr.predict(Xtest)

# Обрахування метрик
print("regr.coef =", np.round(regr.coef_, decimals: 2))
print("regr.intercept =", round(regr.intercept_,  2))
print("R2 score =", round(r2_score(ytest, ypred), 2))
print("Mean absolute error =", round(mean_absolute_error(ytest, ypred), 2))
print("Mean squared error =", round(mean_squared_error(ytest, ypred), 2))

fig, ax = plt.subplots()
ax.scatter(ytest, ypred, edgecolors=(0, 0, 0))
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Виміряно')
ax.set_ylabel('Передбачено')
plt.show()
```

Висновок до завдання: Код демонструє використання лінійної регресії для аналізу та прогнозування даних, а також відображає результати та метрики для оцінки точності моделі.

Завдання 2.5. Самостійна побудова регресії




```
r2:  0.8476571858848433
```

Рис. 8-11: Результат програми та графіки

Код програми:

```python
import numpy as np
from matplotlib import pyplot as plt
from sklearn import linear_model
import sklearn.metrics as sm
from sklearn.preprocessing import PolynomialFeatures

# Генерація даних
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.6 * X ** 2 + X + 2 + np.random.randn(m, 1)
X = X.reshape(-1, 1)
y = y.reshape(-1, 1)


# Лінійна регресія
linear_regressor = linear_model.LinearRegression()
linear_regressor.fit(X, y)

# Поліноміальна регресія
polynomial = PolynomialFeatures(degree=2, include_bias=False)
X_poly = polynomial.fit_transform(X)
polynomial.fit(X_poly, y)

poly_linear_model = linear_model.LinearRegression()
poly_linear_model.fit(X_poly, y)
y_pred = poly_linear_model.predict(X_poly)

print("\nr2: ", sm.r2_score(y, y_pred))
```

```python
# Лінійна регресія
plt.scatter(X, y, color='red')
plt.plot( *args: X, linear_regressor.predict(X), color='blue', linewidth=1)
plt.title("Лінійна регресія")
plt.show()


# Поліноміальна регресія
plt.scatter(X, y, color='red')
plt.plot( *args: X, y_pred, "*", color='blue', linewidth=2)
plt.title("Поліноміальна регресія")
plt.show()
```

Висновок до завдання: Код демонструє використання лінійної та поліноміальної регресії для аналізу та прогнозування даних з квадратичною залежністю. Результати відображаються на графіках для порівняння обох моделей.

Завдання 2.6. Побудова кривих навчання

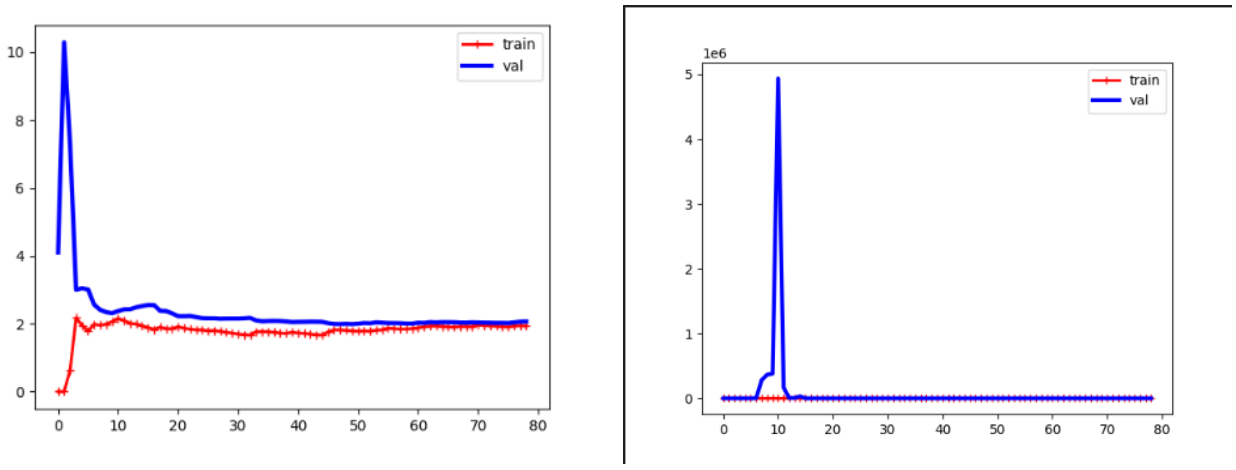


Рис. 12-13: Результат програми та графіки

Код програми:

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

# Генерація даних
m = 100
X = 6 * np.random.rand(m, 1) - 5
y = 0.7 * X ** 2 + X + 3 + np.random.randn(m, 1)


1 usage
def plot_learning_curves(model, X, y):
    X_train, X_val, y_train, y_val = train_test_split( *arrays: X, y, test_size=0.2)
    train_errors, val_errors = [], []
    for m in range(1, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train_predict, y_train[:m]))
        val_errors.append(mean_squared_error(y_val_predict, y_val))
    plt.plot( *args: np.sqrt(train_errors), "r-+", linewidth=2, label='train')
    plt.plot( *args: np.sqrt(val_errors), "b-", linewidth=3, label='val')
    plt.legend()
    plt.show()
```

```python


lin_reg = linear_model.LinearRegression()
# plot_learning_curves(lin_reg, X, y)


polynomial_regression = Pipeline([
    ("poly_features",
     PolynomialFeatures(degree=10, include_bias=False)),
    ("lin_reg", linear_model.LinearRegression())
])

plot_learning_curves(polynomial_regression, X, y)
```

Висновок до завдання: У коді, обрізаному коментарями, функція **plot_learning_curves** була виключена з виконання, але вона може бути активована, щоб побудувати криві навчання і візуалізувати процес навчання моделей.

Завдання 2.7. Кластеризація даних за допомогою методу k-середніх

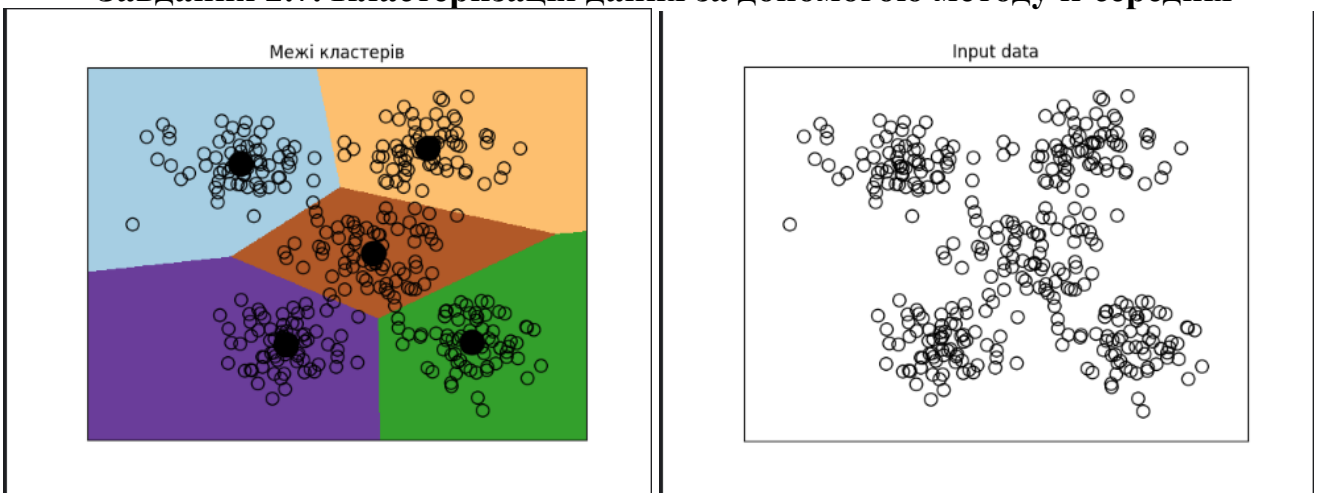


Рис. 14-15: Результат програми та графіки

Код програми:

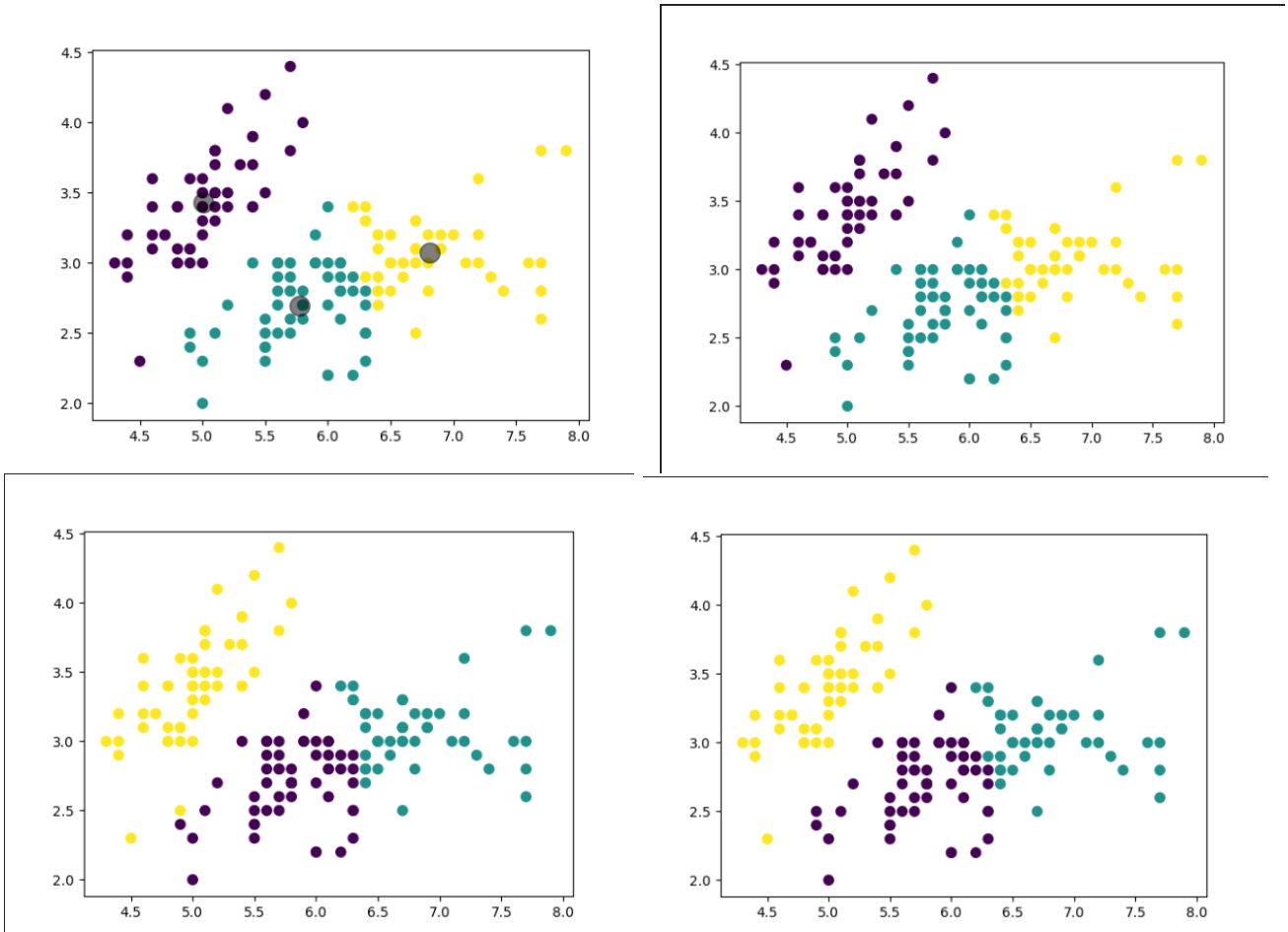
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Завантаження вхідних даних
X = np.loadtxt( fname: 'data_clustering.txt', delimiter=',')
num_clusters = 5

# Включення вхідних даних до графіка
plt.figure()
plt.scatter(X[:, 0], X[:, 1], marker='o', facecolors='none', edgecolors='black', s=80)
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
plt.title('Input data')
plt.xlim( *args: x_min, x_max)
plt.ylim( *args: y_min, y_max)
plt.xticks(())
plt.yticks(())

# Створення об'єкту KMeans
kmeans = KMeans(init='k-means++', n_clusters=num_clusters, n_init=10)

# Навчання моделі кластеризації KMeans
kmeans.fit(X)

# Визначення кроку сітки
step_size = 0.01
```

```python
# Відображення точок сітки
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_vals, y_vals = np.meshgrid( *xi: np.arange(x_min, x_max, step_size),
                             np.arange(y_min, y_max, step_size))

#  Передбачення вихідних міток для всіх точок сітки
output = kmeans.predict(np.c_[x_vals.ravel(), y_vals.ravel()])

# Графічне відображення областей та виділення їх кольором
output = output.reshape(x_vals.shape)
plt.figure()
plt.clf()
plt.imshow(output, interpolation='nearest',
           extent=(x_vals.min(), x_vals.max(),
                   y_vals.min(), y_vals.max()),
           cmap=plt.cm.Paired,
           aspect='auto',
           origin='lower')

# Відображення вхідних точок
plt.scatter(X[:, 0], X[:, 1], marker='o', facecolors='none',
            edgecolors='black', s=80)

# Відображення центрів кластерів
cluster_centers = kmeans.cluster_centers_
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1],
```

```python
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1],
            marker='o', s=210, linewidths=4, color='black',
            zorder=12, facecolors='black')

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
plt.title('Межі кластерів')
plt.xlim( *args: x_min, x_max)
plt.ylim( *args: y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

Висновок до завдання: Результатом цього коду є візуалізація кластеризації вхідних даних за допомогою K-Means, де кожен кластер виділяється окремим кольором, і межі кластерів явно виділені. Ця візуалізація допомагає легше розуміти, як дані розділені на групи за допомогою алгоритму кластеризації.

Завдання 2.8. Кластеризація K-середніх для набору даних Iris



```
Number of clusters: 3, Number of initializations: 10, Maximum iterations: 300, Tolerance: 0.0001, Verbose: 0, Random state: None, Copy data: True
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 1 2 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1
 2 2 2 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 1 2 2 2 1 2 2 2 1 2 2 2 1 2 2
 2 1]
Using find_custom_clusters():
Number of clusters: 3, Random seed: 2
Number of clusters: 3, Random seed: 0
C:\Users\ivanp\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
  super()._check_params_vs_input(X, default_n_init=10)
Number of clusters: 3, Random seed: 0
```

Рис. 16-21: Результат програми та графіки

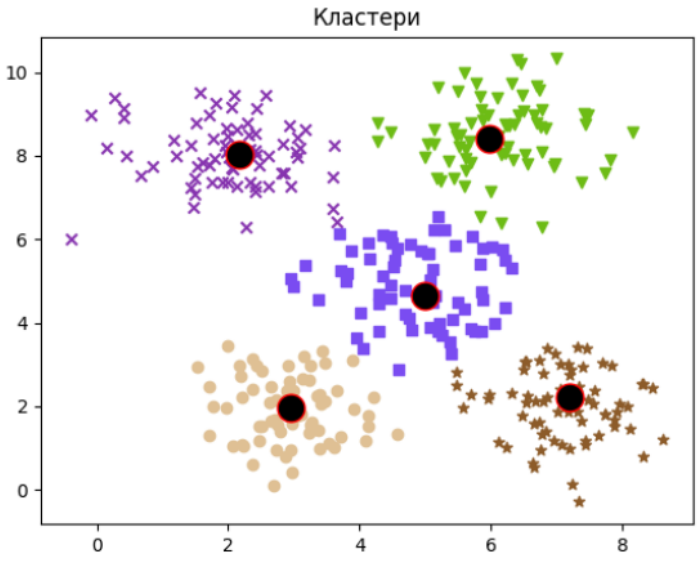
Код програми:

```python
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin
import numpy as np

# Get data
iris = datasets.load_iris()
X = iris.data[:, :2]
Y = iris.target

# Define initial clusters using KMeans
kmeans_clustering = KMeans(n_clusters=Y.max() + 1, init='k-means++', n_init=10, max_iter=300,
                           tol=0.0001, verbose=0, random_state=None, copy_x=True)
kmeans_clustering.fit(X)
predicted_labels = kmeans_clustering.predict(X)

print("Number of clusters: 3, Number of initializations: 10, Maximum iterations: 300, Tolerance: 0.0001, Verbose: 0, "
      "Random state: None, Copy data: True")
print(predicted_labels)
plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=predicted_labels, s=50, cmap='viridis')
cluster_centers = kmeans_clustering.cluster_centers_
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='black', s=200, alpha=0.5)
plt.show()


2 usages
def find_custom_clusters(data, num_clusters, random_seed=2):
    # Randomly select initial cluster centers
    random_generator = np.random.RandomState(random_seed)
    initial_centers_indices = random_generator.permutation(data.shape[0])[:num_clusters]
    custom_centers = data[initial_centers_indices]

    while True:
        # Assign labels based on the nearest center
        custom_labels = pairwise_distances_argmin(data, custom_centers)
        # Find new centers at the mean of the points
        new_custom_centers = np.array([data[custom_labels == i].mean(0) for i in range(num_clusters)])
        # Check for convergence
        if np.all(custom_centers == new_custom_centers):
            break
        custom_centers = new_custom_centers
    return custom_centers, custom_labels


print("Using find_custom_clusters():")
custom_centers, custom_labels = find_custom_clusters(X, num_clusters: 3)
print("Number of clusters: 3, Random seed: 2")
plt.scatter(X[:, 0], X[:, 1], c=custom_labels, s=50, cmap='viridis')
plt.show()

custom_centers, custom_labels = find_custom_clusters(X, num_clusters: 3, random_seed=0)
print("Number of clusters: 3, Random seed: 0")
plt.scatter(X[:, 0], X[:, 1], c=custom_labels, s=50, cmap='viridis')
plt.show()
```

```python
kmeans_labels = KMeans( n_clusters: 3, random_state=0).fit_predict(X)
print("Number of clusters: 3, Random seed: 0")
plt.scatter(X[:, 0], X[:, 1], c=kmeans_labels, s=50, cmap='viridis')
plt.show()
```

Висновок до завдання: Цей код демонструє як використовувати K-Means для кластеризації даних та порівняти його результати з власним методом кластеризації. Візуалізація графіків допомагає зрозуміти різницю у результатах для різних початкових умов.

Завдання 2.9. Оцінка кількості кластерів з використанням методу зсуву середнього




```
Centers of clusters:
 [[2.95568966 1.95775862]
 [7.20690909 2.20836364]
 [2.17603774 8.03283019]
 [5.97960784 8.39078431]
 [4.99466667 4.65844444]]

Number of clusters in input data = 5
```

Рис. 22-23: Результат програми та графіки

Код програми:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MeanShift, estimate_bandwidth

# Завантаження даних
X = np.loadtxt( fname: 'data_clustering.txt', delimiter=',')

# Оцінка ширини вікна для X
bandwidth_X = estimate_bandwidth(X, quantile=0.1, n_samples=len(X))

# Кластеризація даних методом зсуву середнього
meanshift_model = MeanShift(bandwidth=bandwidth_X, bin_seeding=True)
meanshift_model.fit(X)

# Витягування центрів кластерів
cluster_centers = meanshift_model.cluster_centers_
print('\nCenters of clusters:\n', cluster_centers)

# Оцінка кількості кластерів
labels = meanshift_model.labels_
num_clusters = len(np.unique(labels))
print("\nNumber of clusters in input data =", num_clusters)
```

```python
# Відображення на графіку точок та центрів кластерів
plt.figure()
markers = 'o*xvs'
for i, marker in zip(range(num_clusters), markers):
    # Відображення на графіку точок, що належать поточному кластеру
    plt.scatter(X[labels == i, 0], X[labels == i, 1], marker=marker,
                color=np.random.rand(3,))

    # Відображення на графіку центру кластера
    cluster_center = cluster_centers[i]
    plt.plot( *args: cluster_center[0], cluster_center[1], marker='o',
             markerfacecolor='black', markeredgecolor='red',
             markersize=15)

plt.title('Кластери')
plt.show()
```

Висновок до завдання: Mean Shift - це метод без навчання для кластеризації даних, який автоматично визначає кількість кластерів та визначає їх центри на основі розподілу точок у просторі. В результаті отримуємо визначені кластери та їх центри, що може бути корисним для подальшого аналізу даних.

Висновок: використовував спеціалізовані бібліотеки і мову програмування Python щоб дослідити методи регресії та неконтрольованої класифікації даних у машинному навчанні.

Посилання на репозиторій GitHub: https://github.com/IvanPaliy/A.I.-Lab-3-IPZ-Palii.git