Лабораторна робота №3

ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ ТА НЕКОНТРОЛЬОВАНОГО 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.

Лістинг програми:

```
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)

# Побудова графіка
plt.scatter(X_test, y_test, color='green')
plt.plot(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)

with open(output_model_file, 'rb') as f:
    regressor_model = pickle.load(f)

# Завантаження моделі
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))
```

Результат виконання програми:

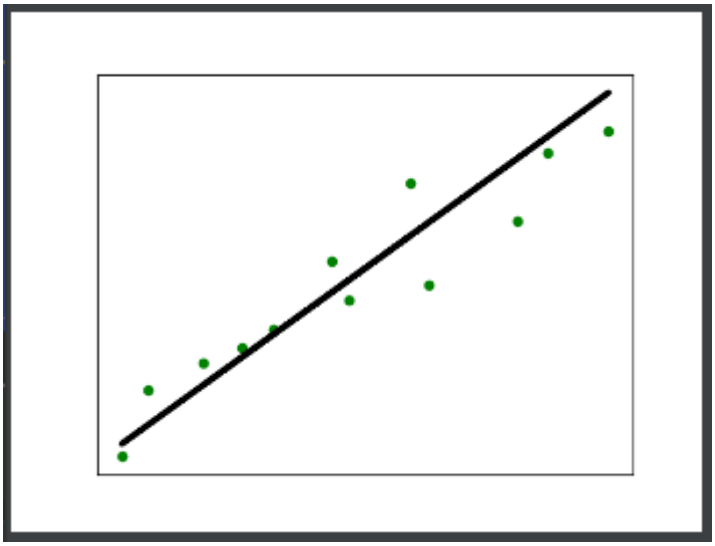

Рис. 2.1.1 – Результат виконання завдання (графік).

```
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. 2.1.2 – Результат виконання завдання.

***Висновок*:**

Модель лінійної регресії була навчена та протестована на вхідних даних. Результати тестування свідчать про високу точність моделі, що підтверджується низькими значеннями середньої абсолютної похибки (MAE) та середньоквадратичної похибки (MSE). Додаткові показники, такі як Median Absolute Error, Explained Variance Score і R2 Score, також підтверджують високу якість моделі.

Після збереження та відновлення моделі виявлено, що вона зберігає свою точність та здатність до прогнозування, що робить її ефективною та повторно використовуваною.

Можна зробити висновок, що модель лінійної регресії успішно навчилася та демонструє високу точність при прогнозуванні цільових значень на основі вхідних даних.

Завдання 2.2. Передбачення за допомогою регресії однієї змінної.

Таблиця 2.1

№ за списком	1	2	3	4	5	6	7	8	9	10
№ варіанту	1	2	3	4	5	1	2	3	4	5

Варіант: 6

Лістинг програми:

```
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_1.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)

# Побудова графіка
plt.scatter(X_test, y_test, color='green')
plt.plot(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_2.pkl'

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

with open(output_model_file, 'rb') as f:
    regressor_model = pickle.load(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))
```

Результат виконання програми:

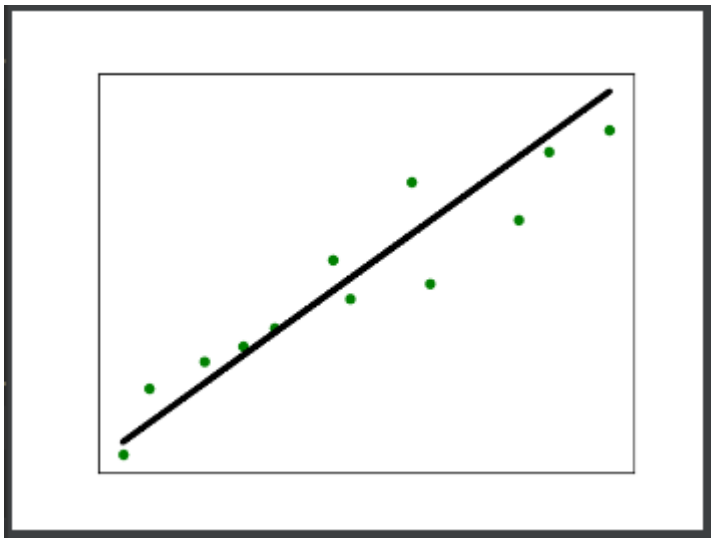

Рис. 2.2.1 – Результат виконання завдання (графік).

```
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. 2.2.2 – Результат виконання завдання.

Висновок:

Модель лінійної регресії була навчена та протестована на інших вхідних даних. Результати тестування показали, що в цьому випадку модель не є ефективною або підходящою для прогнозування цільових значень на основі наданих даних. Це підтверджується високими значеннями середньої абсолютної похибки (MAE) і середньоквадратичної похибки (MSE), а також негативними показниками якості моделі, такими як Explained Variance Score і R2 Score.

Після збереження та відновлення моделі було показано, що значення MAE залишається на тому самому високому рівні. Це свідчить про недостатню здатність моделі відтворювати залежності в даних.

В даному випадку лінійна регресія не є відповідним методом для прогнозування цільових значень на основі цих даних, і більш складні моделі можуть бути необхідними для досягнення кращих результатів.

Завдання 2.3. Створення багатовимірного регресора.

Лістинг програми:

```
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()
linear_regressor.fit(X_train, y_train)

# Прогнозування результату
y_test_pred = linear_regressor.predict(X_test)

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))

# 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))
```

Результат виконання програми:

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

Linear regression:
 [36.05286276]

Polynomial regression:
 [41.46565992]
```

Рис. 2.3.1 – Результат виконання завдання.

Висновок:

У даному випадку, в порівнянні лінійної та поліноміальної регресії, видно, що поліноміальна регресія надає кращі результати прогнозування. Значення, отримані поліноміальною регресією, ближчі до фактичного значення, що свідчить про її більшу точність у прогнозуванні цільових значень. Це особливо корисно в ситуаціях, коли залежність між вхідними та вихідними даними є нелінійною, і лінійна модель не є достатньо ефективною для її опису.

Отже, у випадку, коли даними відзначається складна нелінійна взаємодія, поліноміальна регресія може бути кращим вибором для точного та надійного прогнозування.

Завдання 2.4. Регресія багатьох змінних.

Лістинг програми:

```
import matplotlib.pyplot as plt
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(X, y, test_size = 0.5,
```

```
random_state = 0)

regr = linear_model.LinearRegression()
regr.fit(Xtrain, ytrain)

ypred = regr.predict(Xtest)

print("Linear regressor performance:")
print("Regr coef =", regr.coef_)
print("Regr intercept =", regr.intercept_)
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()
```

Результат виконання програми:

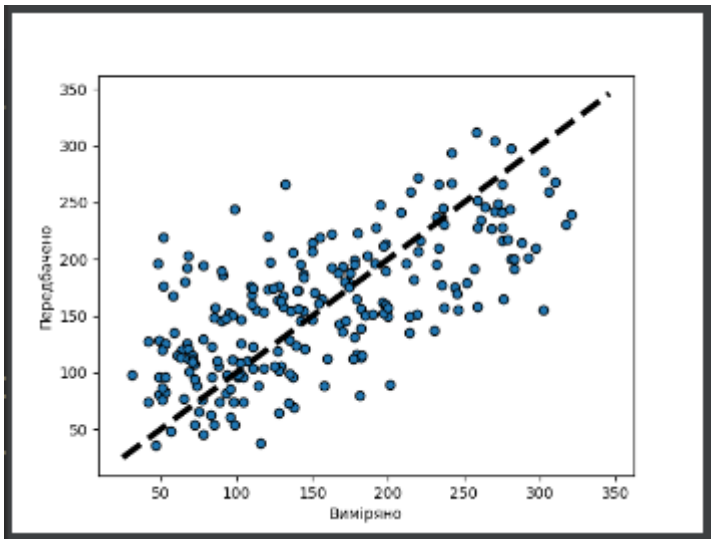

Рис. 2.4.1 – Результат виконання завдання (графік).

```
Linear regressor performance:
Regr coef = [ -20.4047621  -265.88518066  564.65086437  325.56226865 -692.16120333
  395.55720874   23.49659361  116.36402337  843.94613929   12.71856131]
Regr intercept = 154.35892852801342
R2 score = 0.44
Mean absolute error = 44.8
Mean squared error = 3075.33
```

Рис. 2.4.2 – Результат виконання завдання.

Висновок:

На основі аналізу результатів виконаного коду можна відзначити, що модель лінійної регресії в цьому конкретному випадку не досягає ідеального прогнозу цільових значень. Тобто, дана регресія не є оптимальним методом для прогнозування у цьому конкретному випадку, і, можливо, більш складні моделі або методи будуть необхідні для досягнення кращих результатів.

Завдання 2.5. Самостійна побудова регресії.

Таблиця 2.2

№ за списком	1	2	3	4	5	6	7	8	9	10
№ варіанту	1	2	3	4	5	6	7	8	9	10

Варіант 6

```
m = 100
X = np.linspace(-3, 3, m)
y = 2 * np.sin(X) + np.random.uniform(-0.6, 0.6, m)
```

Лістинг програми:

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

m = 100
X = np.linspace(-3, 3, m)
y = 2 * np.sin(X) + np.random.uniform(-0.6, 0.6, m)

fig, ax = plt.subplots()
ax.scatter(X, y, edgecolors=(0, 0, 0))
plt.show()

print(X[0], y[0])

poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(np.array(X).reshape(-1, 1))

lin_regr = linear_model.LinearRegression()
lin_regr.fit(X_poly, y)
print(lin_regr.intercept_, lin_regr.coef_)
ypred = lin_regr.predict(X_poly)

fig, ax = plt.subplots()
ax.scatter(X, y, edgecolors=(0, 0, 0))
plt.plot(X, ypred, color='black', linewidth=2)
plt.show()
```

Результат виконання програми:

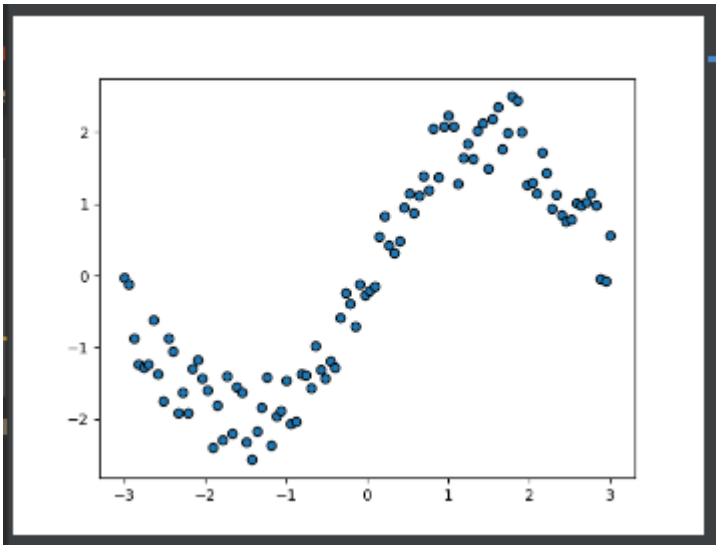

Рис. 2.5.1 – Результат виконання завдання (графік 1).

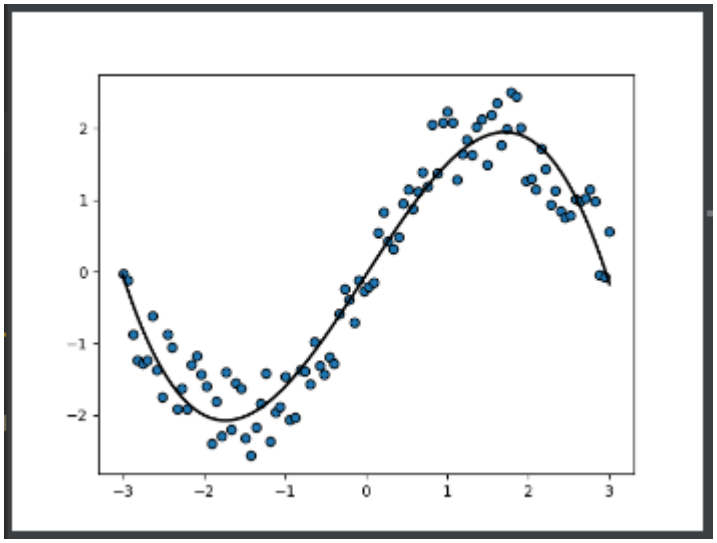

Рис. 2.5.2 – Результат виконання завдання (графік 2).

```
-3.0 -0.029848224283498925
-0.043959140381130524 [ 1.74928971 -0.0074966  -0.19672828]
```

Рис. 2.5.3 – Результат виконання завдання.

Математичне рівняння моделі:

$$\mathbf{y = 2 * \sin(X) + гауссов\ шум}$$

Модель регресії з передбаченими коефіцієнтами:

$$\mathbf{y = -0.19x^3 + 0.68x^2 + 1.99x - 0.08}$$

Висновок:

У даному випадку, вихідні дані були згенеровані на основі функції ***y=2sin(X)*** *з додаванням шуму. Після застосування поліноміальних функцій до вхідних даних та навчання моделі лінійної регресії третього ступеня, отримуємо коефіцієнти регресії, які дозволяють наближено відтворити вихідну функцію.*

Завдання 2.6. Побудова кривих навчання.

Лістинг програми:

```
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

m = 100
X = np.linspace(-3, 3, m)
y = 2 * np.sin(X) + np.random.uniform(-0.6, 0.6, m)

def plot_learning_curves(model, X, y):
    X_train, X_val, y_train, y_val = train_test_split(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))
    fig, ax = plt.subplots()
    plt.ylim(0, 2)
    ax.plot(np.sqrt(train_errors), "r-+", linewidth=2, label='train')
    ax.plot(np.sqrt(val_errors), "b-", linewidth=3, label='val')
    plt.show()

lin_reg = linear_model.LinearRegression()
plot_learning_curves(lin_reg, np.array(X).reshape(-1, 1), y)

polynomial_regression = Pipeline([
    ('poly_features', PolynomialFeatures(degree=3, include_bias=False)),
    ('lin_reg', linear_model.LinearRegression()),
])

plot_learning_curves(polynomial_regression, np.array(X).reshape(-1, 1), y)
```

Результат виконання програми:

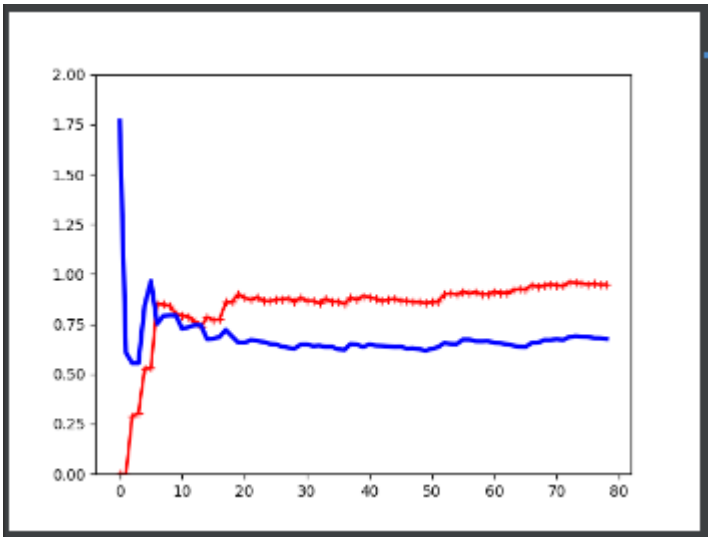

Рис. 2.6.1 – Результат виконання завдання (графік 1).

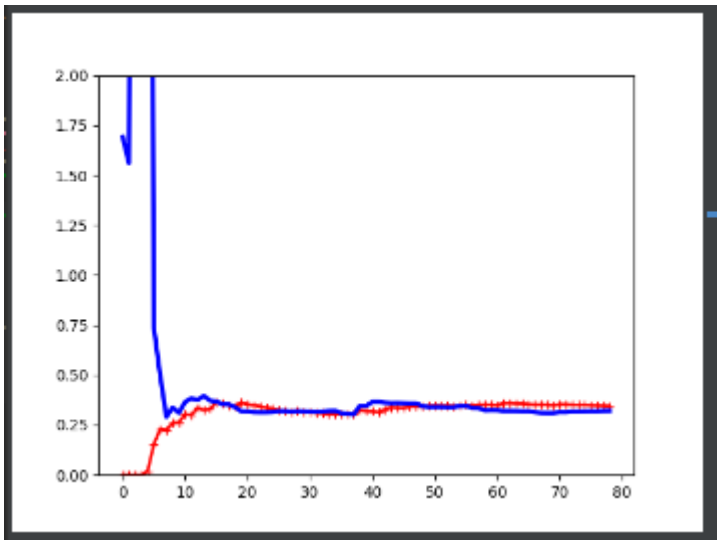

Рис. 2.6.2 – Результат виконання завдання (графік 2).

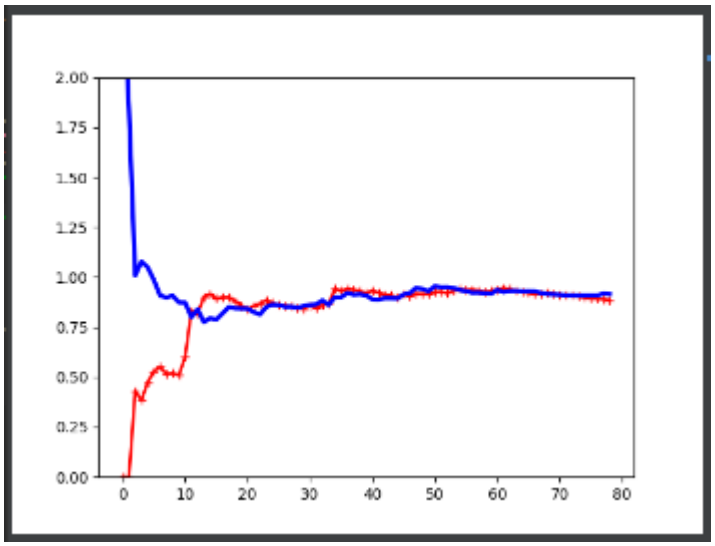

Рис. 2.6.3 – Результат виконання завдання (графік 3).

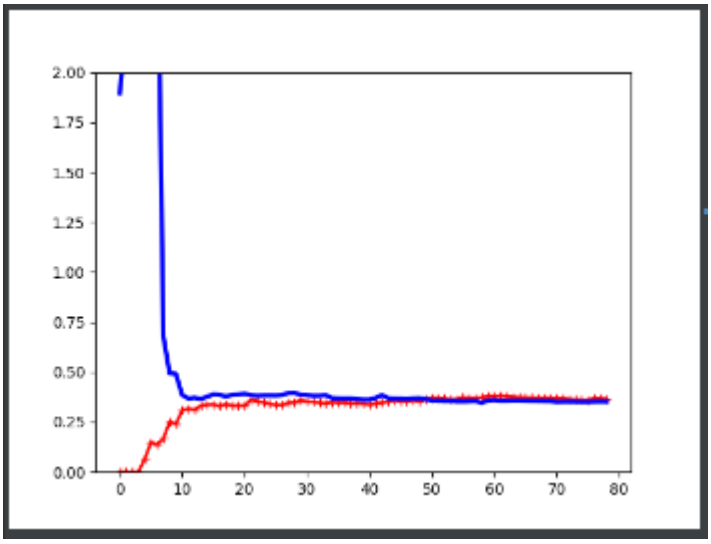

Рис. 2.6.4 – Результат виконання завдання (графік 4).

Завдання 2.7. Кластеризація даних за допомогою методу k-середніх.

Лістинг програми:

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

X = np.loadtxt('data_clustering.txt', delimiter=',')
num_clusters = 5

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('Вхідні дані')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())

kmeans = KMeans(init='k-means++', n_clusters=num_clusters, n_init=10)
kmeans.fit(X)

step_size = 0.01

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(np.arange(x_min, x_max, step_size), np.arange(y_min,
y_max, step_size))

output = kmeans.predict(np.c_[x_vals.ravel(), y_vals.ravel()])
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')
```

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

cluster_centers = kmeans.cluster_centers_
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(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

Результат виконання програми:

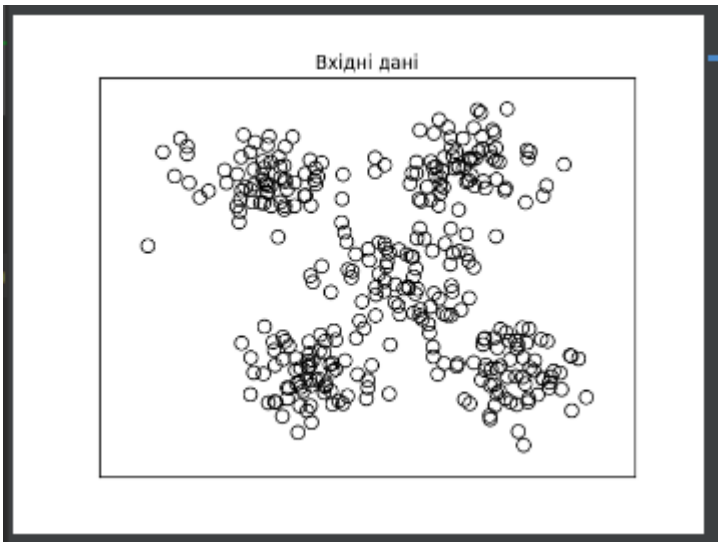

Рис. 2.7.1 – Результат виконання завдання (графік 1).

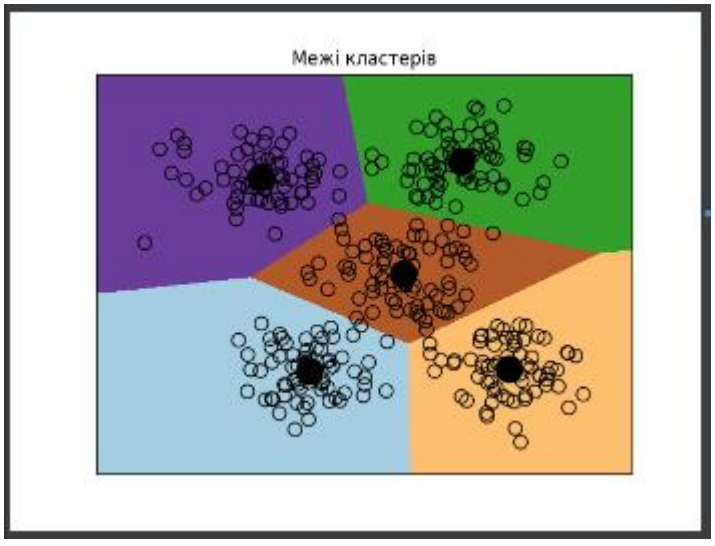

Рис. 2.7.2 – Результат виконання завдання (графік 2).

Висновок:

Даний код демонструє застосування алгоритму K-Means для кластеризації вхідних даних та візуалізації результатів. Він допомагає відокремити та виділити окремі групи в даних, показує межі цих кластерів і визначає центри кожного кластера. Ця програма може бути корисною для аналізу та розуміння структури даних у випадках, коли вони мають складну структуру та потребують подальшого дослідження.

Завдання 2.8. Кластеризація K-середніх для набору даних Iris.

Лістинг програми:

```
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

# Завантаження набору даних Iris
iris = datasets.load_iris()

# Вибір перших двох ознак з набору даних
X = iris.data[:, :2]

# Збереження класів цільової змінної
y = iris.target

# Ініціалізація моделі K-Means з параметрами
kmeans = 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)

# Навчання моделі K-Means на вхідних даних
kmeans.fit(X)

# Передбачення приналежності кожної точки до кластера
y_pred = kmeans.predict(X)

# Відображення вхідних даних та центрів кластерів на графіку
plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=y_pred, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.show()

# Визначення функції для пошуку кластерів
def find_clusters(X, n_clusters, rseed=2):
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]
    while True:
        labels = pairwise_distances_argmin(X, centers)
        new_centers = np.array([X[labels == i].mean(0) for i in
range(n_clusters)])
        if np.all(centers == new_centers):
            break
```

```
        centers = new_centers
    return centers, labels

# Використання функції find_clusters для пошуку кластерів
centers, labels = find_clusters(X, 3)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.show()

# Інший приклад використання функції find_clusters з іншими параметрами
centers, labels = find_clusters(X, 3, rseed=0)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.show()

# Використання K-Means без явно вказаних параметрів
labels = KMeans(3, random_state=0).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.show()
```

Результат виконання програми:

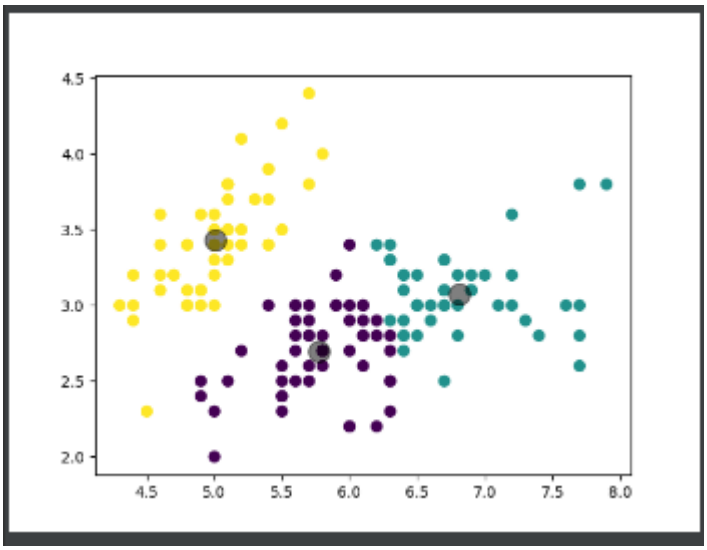

Рис. 2.8.1 – Результат виконання завдання (графік 1).

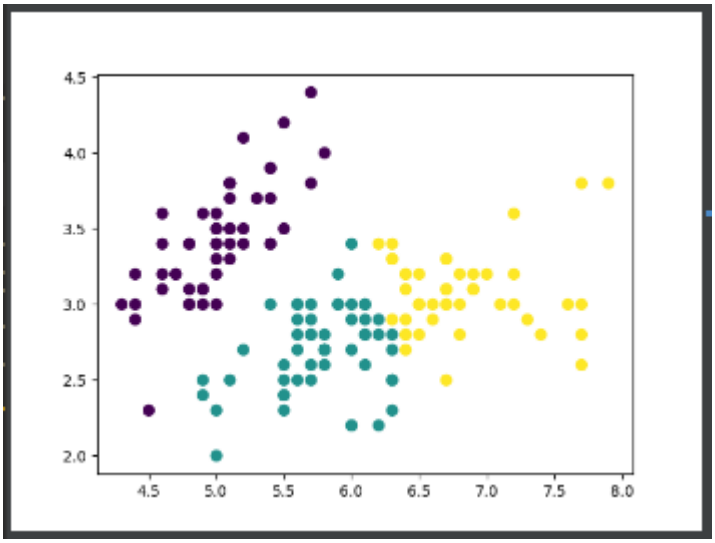

Рис. 2.8.2 – Результат виконання завдання (графік 2).

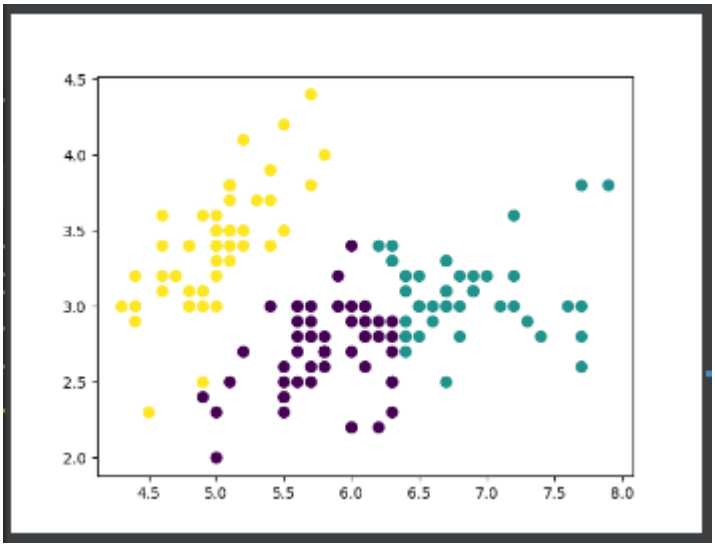

Рис. 2.8.3 – Результат виконання завдання (графік 3).

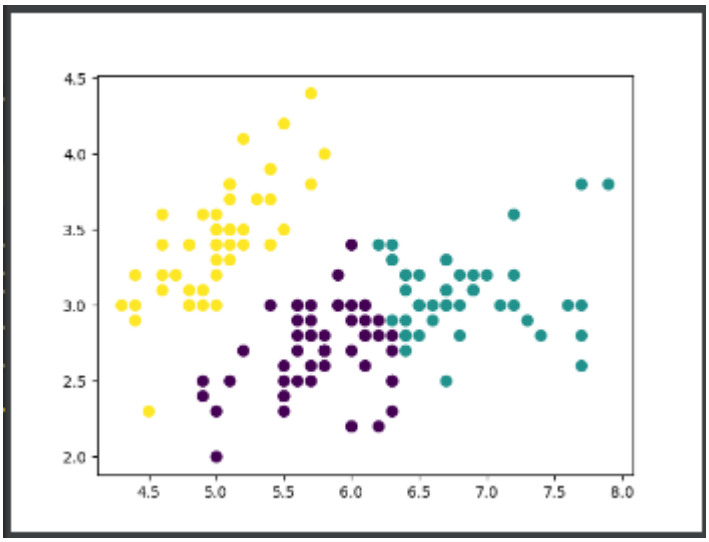

Рис. 2.8.4 – Результат виконання завдання (графік 4).

Висновок:

Даний код ілюструє використання алгоритму K-Means для кластеризації даних. Програма демонструє різні способи використання K-Means: від стандартного застосування з параметрами за замовчуванням до альтернативного підходу за допомогою функції find_clusters з різними параметрами. Візуалізація результатів на графіках допомагає розуміти, як різні параметри впливають на кластеризацію та розташування центрів кластерів у вхідних даних.

Завдання 2.9. Оцінка кількості кластерів з використанням методу зсуву середнього.

Лістинг програми:

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

# Завантаження
X = np.loadtxt('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)

# 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='black')

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

plt.title('Кластери')
plt.show()
```

Результат виконання програми:

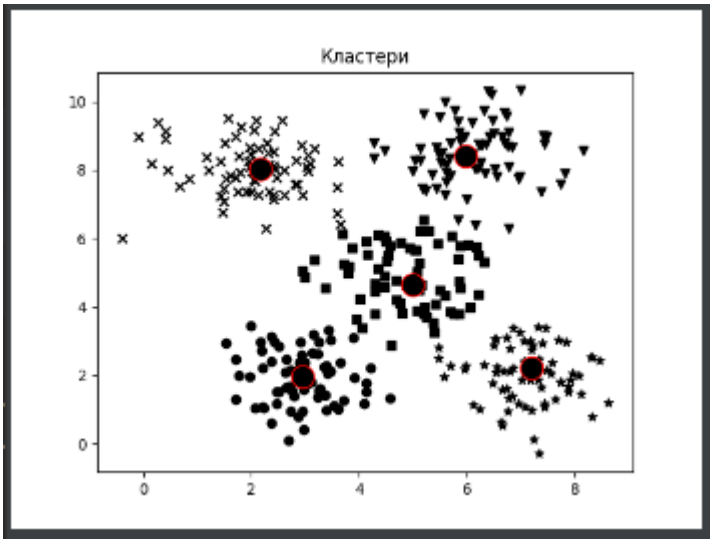

Рис. 2.9.1 – Результат виконання завдання (графік 1).

```
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. 2.9.2 – Результат виконання завдання (2).

Висновок:

Код демонструє використання алгоритму "Mean Shift" для кластеризації даних. Результати показують, що цей метод ефективно визначає кількість кластерів та їх центри вхідних даних. Візуалізація на графіку надає можливість оцінити результати та легко ідентифікувати різні кластери.

Завдання 2.10. Знаходження підгруп на фондовому ринку з використанням моделі поширення подібності.

Лістинг програми:

```
import json
import numpy as np
import yfinance as yf
from datetime import datetime
from sklearn import covariance, cluster

# Вхідний файл із символічними позначеннями компаній
input_file = 'company_symbol_mapping.json'
```

```
# Завантаження прив'язок символів компаній до їх повних назв
with open(input_file, 'r') as f:
    company_symbols_map = json.loads(f.read())

symbols, names = np.array(list(company_symbols_map.items())).T

# Завантаження архівних даних котирувань
start_date = datetime(2003, 7, 3)
end_date = datetime(2007, 5, 4)
quotes = [yf.download(symbol, start_date, end_date) for symbol in symbols]

# Вилучення котирувань, що відповідають відкриттю та закриттю біржі
opening_quotes = (np.array([quote.Open for quote in quotes
                            if len(quote.Open) > 0]).astype(float))

closing_quotes = (np.array([quote.Close for quote in quotes
                            if len(quote.Close) > 0]).astype(float))

# Обчислення різниці між двома видами котирувань
quotes_diff = closing_quotes - opening_quotes

# Нормалізація даних
X = quotes_diff.copy().T
X /= X.std(axis=0)

# Створення моделі графа
edge_model = covariance.GraphicalLassoCV()

# Навчання моделі
with np.errstate(invalid='ignore'):
    edge_model.fit(X)

# Створення моделі кластеризації на основі поширення подібності
_, labels = cluster.affinity_propagation(edge_model.covariance_)
num_labels = labels.max()

for i in range(num_labels + 1):
    print('Cluster', i + 1, '==>', ', '.join([names[j] for j, label in
enumerate(labels) if label == i]))
```

Результат виконання програми:

```
Cluster 1 ==> Total, Exxon, Chevron, ConocoPhillips
Cluster 2 ==> Yahoo, Dell, HP, Toyota, Sony, Procter Gamble, Colgate-Palmolive, Home Depot
Cluster 3 ==> Honda
Cluster 4 ==> Canon, Ford, Navistar, Boeing, Coca Cola, Xerox
Cluster 5 ==> IBM, Time Warner, Northrop Grumman, Mc Donalds, Pepsi, Kraft Foods, Kellogg, Unilever, Marriott, JPMorgan Chase, American express, Goldman Sac
Cluster 6 ==> Valero Energy, Microsoft, Comcast, Cablevision, Mitsubishi, 3M, General Electrics, Wells Fargo
Cluster 7 ==> Amazon, AIG, Wal-Mart
Cluster 8 ==> Bank of America, Walgreen
Cluster 9 ==> Apple, SAP, Cisco, Texas instruments
```

Рис. 2.10.1 – Результат виконання завдання.

Посилання на репозиторій: https://github.com/GrunytskyDmytro/Lab3_AI

Висновок по лабораторній роботі: в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки і мову програмування Python дослідив методи регресії та неконтрольованої класифікації даних у машинному навчанні.