ЛАБОРАТОРНА РОБОТА № 3

Тема: **«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**

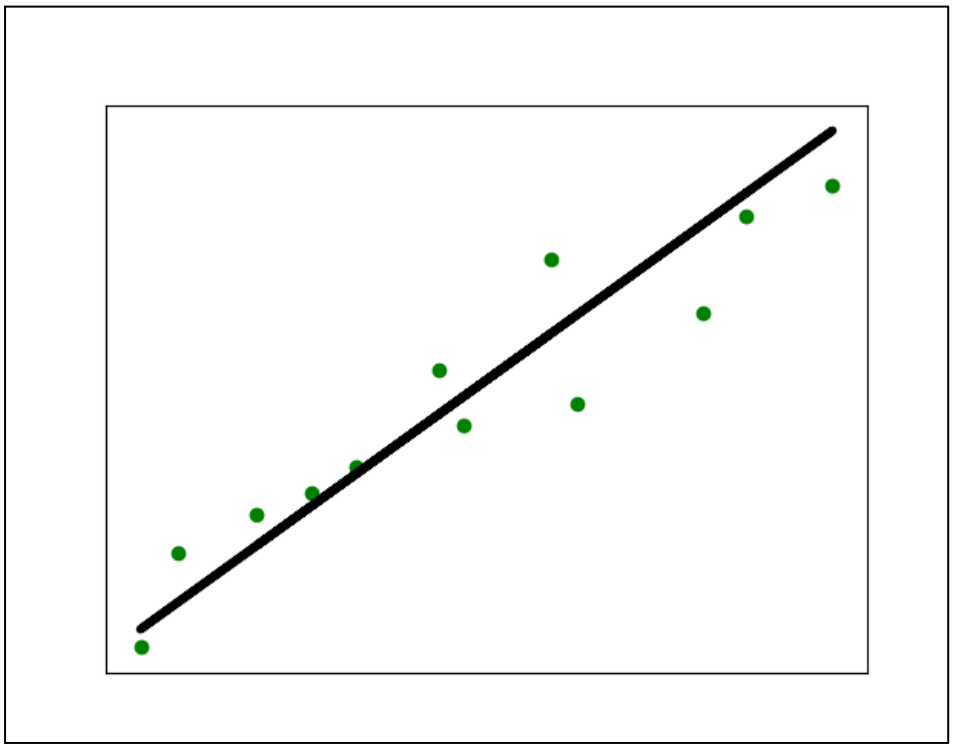
Побудувати регресійну модель на основі однієї змінної. Використовувати файл вхідних даних: data_singlevar_regr.txt.

Лістинг програми:

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

input_file = 'data_singlevar_regr.txt'

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

num_training = int(0.8 * len(X))
num_test = len(X) - num_training

X_train, y_train = X[:num_training], y[:num_training]

X_test, y_test = X[num_training:], y[num_training:]

regressor = linear_model.LinearRegression()

regressor.fit(X_train, y_train)

y_test_pred = regressor.predict(X_test)
```

```
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))

output_model_file = 'model.pkl'

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)

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:



```
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. 1.1. – 1.2. Результат виконання програми

Висновки щодо результатів оцінки якості

Регресійна модель має досить високу точність, що підтверджується високим значенням коефіцієнту детермінації (0,86) та оцінки дисперсії (0,86). Отже, модель добре справляється з поставленим завданням та немає великих викидів, згідно з графіком та даних отримних з MAE та MSE.

Завдання 2.2. **Передбачення за допомогою регресії однієї змінної**

Побудувати регресійну модель на основі однієї змінної. Використовувати вхідні дані відповідно свого варіанту, що визначається за списком групи у журналі (таблиця 2.1).

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

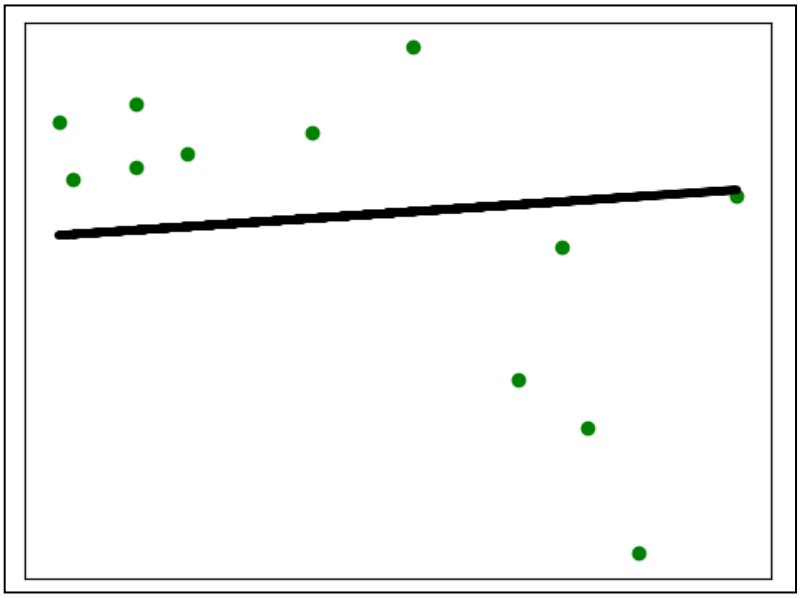
Файл для 5 варіанту: data_regr_5.txt

Лістинг програми:

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

input_file = 'data_regr_5.txt'

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

num_training = int(0.8 * len(X))
num_test = len(X) - num_training
```

```
X_train, y_train = X[:num_training], y[:num_training]

X_test, y_test = X[num_training:], y[num_training:]

regressor = linear_model.LinearRegression()

regressor.fit(X_train, y_train)

y_test_pred = regressor.predict(X_test)

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))

output_model_file = 'model2.pkl'

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)

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:



```
Linear regressor performance:
Mean absolute error = 3.31
Mean squared error = 16.98
Median absolute error = 2.66
Explain variance score = -0.14
R2 score = -0.15

New mean absolute error = 3.31
```

Рис. 2.1. – 2.2. Результат виконання програми

Висновки щодо результатів оцінки якості

З отриманими метриками можна зробити висновок, що модель показує низьку точність у передбаченні вихідних даних. Зокрема, середня абсолютна похибка складає 3.31, що означає, що в середньому модель відхиляється на цю величину від справжніх значень. Крім того, оцінка дисперсії та коефіцієнт детермінації негативні (-0.14 та -0.15 відповідно), що свідчить про те, що модель не в змозі ефективно враховувати різноманітність вихідних даних. Також бачимо, що в датасеті є аномальні дані, до яких модель не пристосована. Отже, на другому наборі даних модель показала себе значно гірше, ніж на першому.

Завдання 2.3. **Створення багатовимірного регресора**

Використовувати файл вхідних даних: data_multivar_regr.txt, побудувати регресійну модель на основі багатьох змінних.

Лістинг програми:

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

input_file = 'data_multivar_regr.txt'

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

num_training = int(0.8 * len(X))
num_test = len(X) - num_training
```

```
X_train, y_train = X[:num_training], y[:num_training]

X_test, y_test = X[num_training:], y[num_training:]

linear_regressor = linear_model.LinearRegression()

linear_regressor.fit(X_train, y_train)

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("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))

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
Explained variance score = 0.86
R2 score = 0.86

Linear regression:
 [36.05286276]

Polynomial regression:
 [41.45976677]
```

Рис. 3. Результат виконання програми

Висновки щодо оцінки та порівняння отриманих характеристик

Отже, аналізуючи отримані результати, можна дійти висновку, що поліноміальний регресор краще справляється за лінійний регресор при регресії з декількома характеристиками. Він забезпечує отримання результату, ближчого до значення 41.45, тобто дає кращі результати.

Завдання 2.4. Регресія багатьох змінних

Розробіть лінійний регресор, використовуючи набір даних по діабету, який існує в sklearn.datasets.

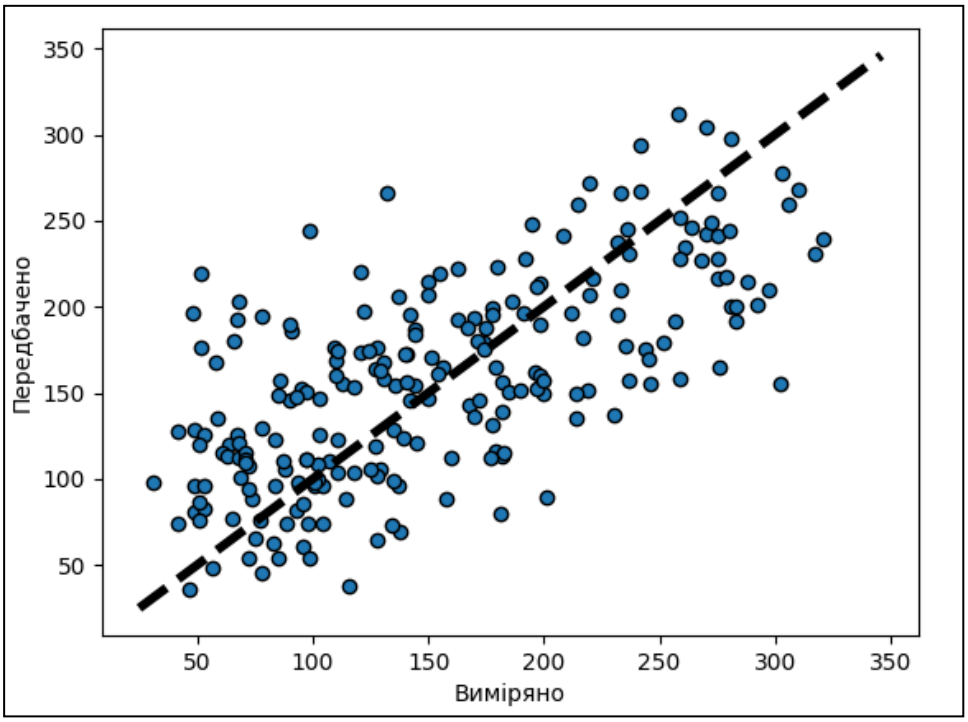
Набір даних містить 10 вихідних змінних — вік, стать, індекс маси тіла, середній артеріальний тиск і шість вимірювань сироватки крові, отриманих у 442 пацієнтів із цукровим діабетом, а також реакцію, що цікавить, — кількісний показник прогресування захворювання через 1 рік після вихідного рівня. Отже, існує 442 екземпляри з 10 атрибутами. Колонка 11 є кількісною мірою прогресування захворювання через 1 рік після вихідного рівня. Кожен з цих 10 атрибутів був відцентрований по середньому та масштабований за часом стандартного відхилення n_samples (тобто сума квадратів кожного стовпця складає 1).

Лістинг програми:

```
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(X, y, test_size=0.5, ran-
dom_state=0)
regr = linear_model.LinearRegression()
regr.fit(Xtrain, ytrain)
ypred = regr.predict(Xtest)

print("regr.coef =", np.round(regr.coef_, 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()
```

Результат виконання програми:




```
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. 4.1 – 4.2. Результат виконання програми

Висновки щодо оцінки та порівняння отриманих характеристик

Отже, згідно з графіком та отриманими результатами, можна дійти висновку, що дані сильно розкидані, а похибка є відносно великою. Значення коефіцієнту детермінації є досить низьким для точної регресійної моделі — 0,44. Тому модель не може ефективно пояснювати зміну цукру в крові на основі цих ознак, особливо на великій кількості даних.

Завдання 2.5. Самостійна побудова регресії

Згенеруйте свої випадкові дані обравши за списком відповідно свій варіант (згідно табл. 2.2) та виведіть їх на графік. Побудуйте по них модель лінійної регресії, виведіть на графік. Побудуйте по них модель поліноміальної регресії, виведіть на графік. Оцініть її якість.

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 5

```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.4 * X ** 2 + X + 4 + np.random.randn(m, 1)
```

Лістинг програми:

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

m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.4 * X ** 2 + X + 4 + np.random.randn(m, 1)

fig, ax = plt.subplots()
ax.scatter(X, y, edgecolors=(0, 0, 0))
plt.title("Лінійна регресія")
plt.show()

print(X[1], y[1])

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

linear_regression = linear_model.LinearRegression()
linear_regression.fit(X_poly, y)
print(linear_regression.intercept_, linear_regression.coef_)
y_pred = linear_regression.predict(X_poly)

fig, ax = plt.subplots()
ax.scatter(X, y, edgecolors=(0, 0, 0))
plt.plot(X, y_pred, color='red', linewidth=4)
plt.title("Поліноміальна регресія")
plt.show()
```

Результат виконання програми:

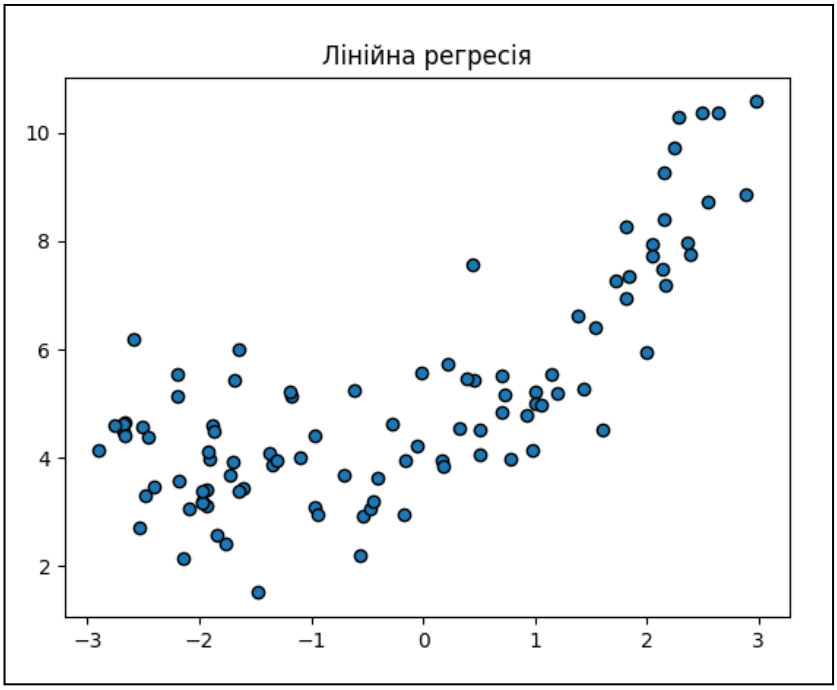


Рис. 5.1. Результат виконання програми (лінійна регресія)

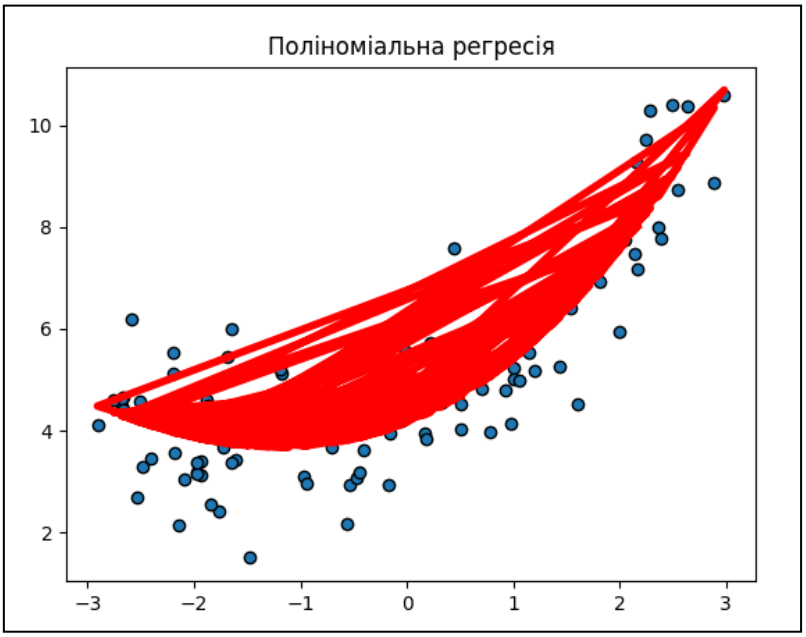


Рис. 5.2. Результат виконання програми (поліноміальна регресія)

```
[-1.47964278] [2.07862495]
[4.19268448] [[ 0.97432551  0.33127937 -0.00840903]]
```

Рис. 5.3. Результат виконання програми

Модель у вигляді математичного рівняння

Модель: $y = 0.4x^2 + x + 4 + шум\ гауса$

Отримана модель регресії з передбаченими коефіцієнтами:

$0.97x^2 + 0.33x - 0.008$

Висновки щодо оцінки якості

Отже, можна зробити висновок, що поліноміальна регресія дає змогу будувати моделі для нелінійних даних і при цьому забезпечує чудовий результат. Можна сказати, що модель добре адаптується до даних і має високу прогностичну здатність.

Завдання 2.6. **Побудова кривих навчання**

Побудуйте криві навчання для ваших даних у попередньому завданні.

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 = 6 * np.random.rand(m, 1) - 3
y = 0.4 * X ** 2 + X + 4 + np.random.randn(m, 1)


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))
    plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label='train')
    plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label='val')
    plt.legend()
    plt.show()


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

```
polynomial_regression_degree10 = Pipeline([
    ("poly_features",
     PolynomialFeatures(degree=10, include_bias=False)),
    ("lin_reg", linear_model.LinearRegression())
])

plot_learning_curves(polynomial_regression_degree10, X, y)

polynomial_regression_degree2 = Pipeline([
    ("poly_features",
     PolynomialFeatures(degree=2, include_bias=False)),
    ("lin_reg", linear_model.LinearRegression())
])

plot_learning_curves(polynomial_regression_degree2, X, y)
```

Результат виконання програми:

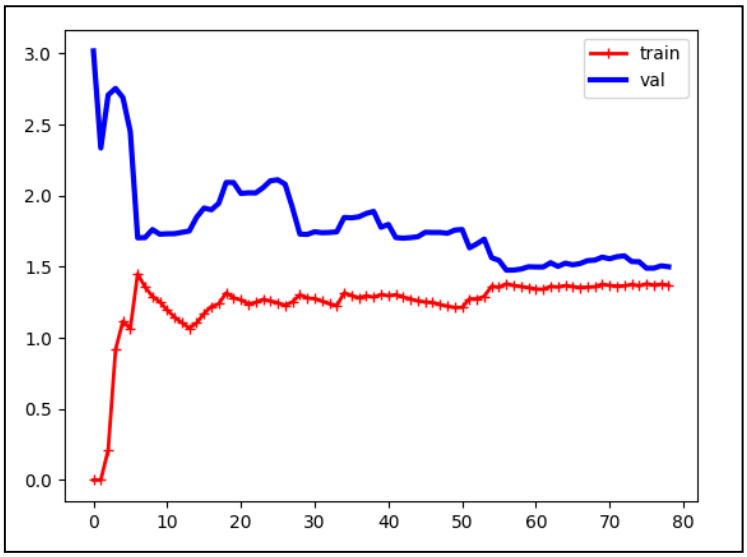


Рис. 6.1. Криві навчання для лінійної моделі

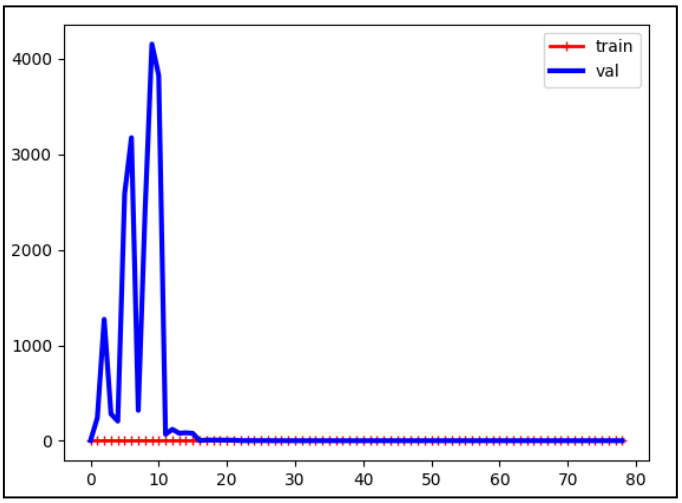


Рис. 6.2. Криві навчання для поліноміальної моделі 10-го ступеня

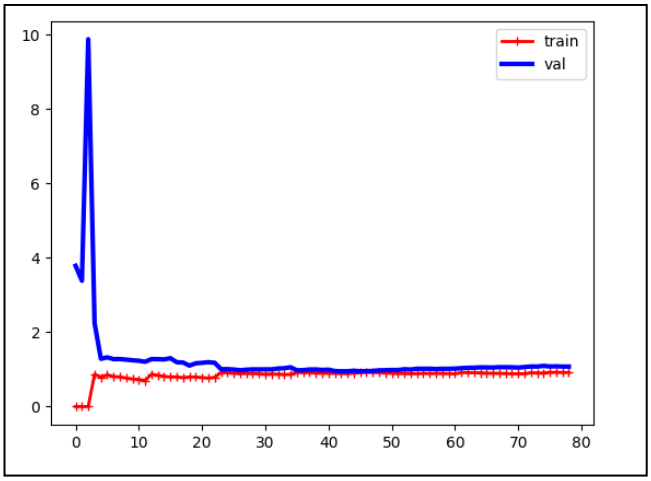


Рис. 6.3. Криві навчання для поліноміальної моделі 2-гоступеня

Висновок до завдання

Для визначення оптимального рівня складності моделі, ми використали криві навчання. Це дозволило нам зрозуміти, чи варто використовувати більш складну чи просту модель для розв'язання даної проблеми. В результаті аналізу ми прийшли до висновку, що модель 2-го ступеня найкраще вирішує поставлену задачу. Це підтверджено тим, що криві навчання для цієї моделі показали зближення навчального та тестового наборів даних до високого рівня точності. Це означає, що модель 2-го ступеня виявилася досить складною для адекватного відтворення даних, але водночас не надто складною, щоб виникла проблема перенавчання. Таким чином, це може бути оптимальним компромісом між точністю та загальною здатністю до узагальнення на нові дані.

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

Провести кластеризацію даних методом k-середніх. Використовувати файл вхідних даних: data_clustering.txt.

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('Input data')
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:

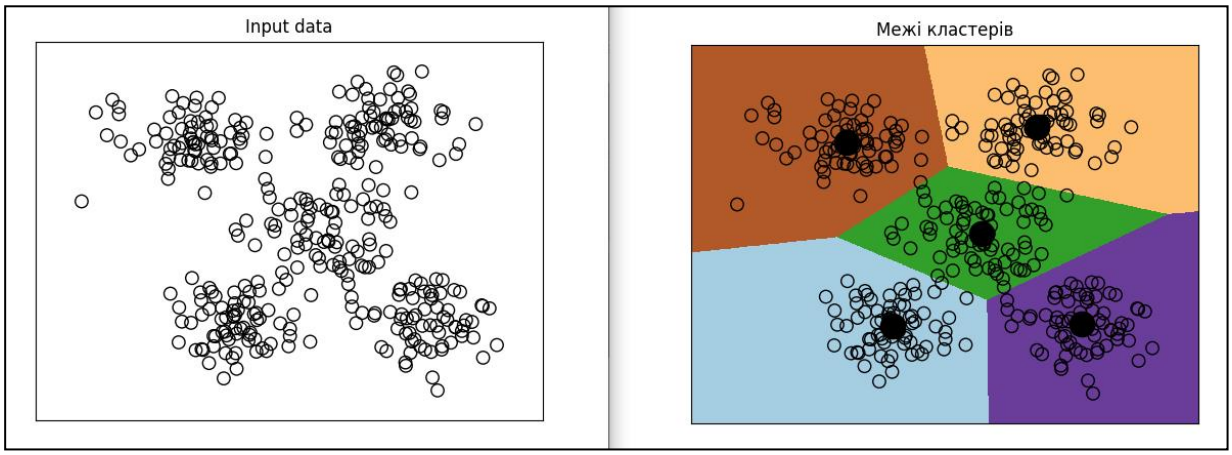


Рис. 7. Результат виконання програми

Висновок до завдання

Метод k-середніх є ефективним алгоритмом для кластеризації даних без учителя. Проте, важливо зазначити, що цей метод передбачає заздалегідь відому кількість кластерів, що може бути важко визначити в певних ситуаціях.

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

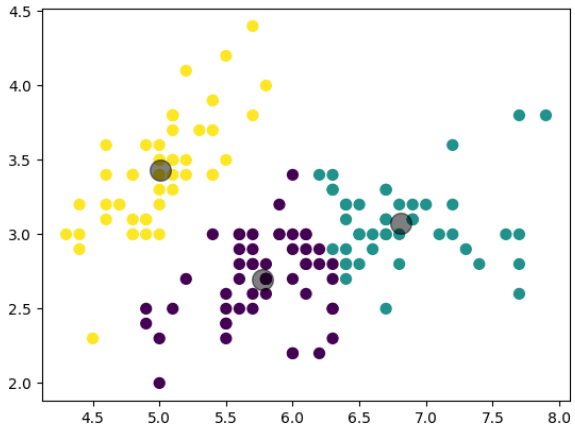
Виконайте кластеризацію K-середніх для набору даних Iris, який включає три типи (класи) квітів ірису (Setosa, Versicolour і Virginica) з чотирма атрибутами: довжина чашолистка, ширина чашолистка, довжина пелюстки та ширина пелюстки. У цьому завданні використовуйте sklearn.cluster.KMeans для пошуку кластерів набору даних Iris.

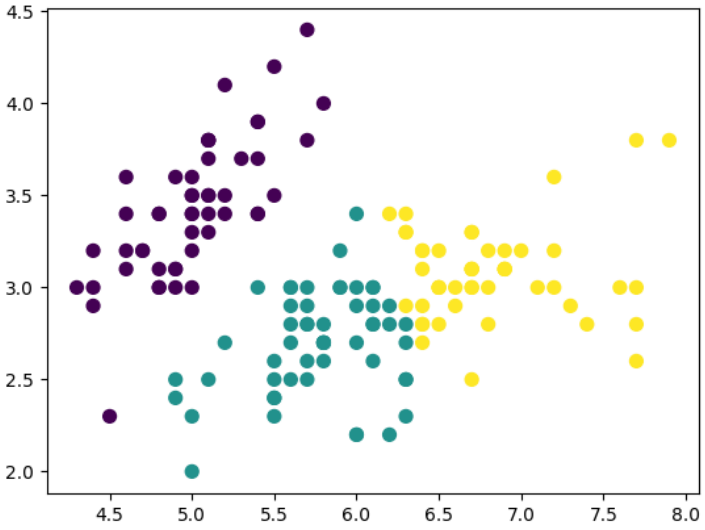
Лістинг програми:

```
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

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

# Визначаємо початкові кластери
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)
kmeans.fit(X)
y_pred = kmeans.predict(X)

print("n_clusters: 3, n_init: 10, max_iter: 300, tol: 0.0001, verbose: 0, ran-
```

```
dom_state: None, copy_x: True")
print(y_pred)
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()


def find_clusters(X, n_clusters, rseed=2):
    # Випадково обираємо кластери
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]

    while True:
        # Оголошуємо label базуючись на найближчому центрі
        labels = pairwise_distances_argmin(X, centers)
        # Знаходимо нові центри з середини точок
        new_centers = np.array([X[labels == i].mean(0) for i in
range(n_clusters)])
        # Перевірка збіжності
        if np.all(centers == new_centers):
            break
        centers = new_centers
    return centers, labels


print("using find_clusters():")
centers, labels = find_clusters(X, 3)
print("n_clusters: 3, rseed: 2")
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.show()

centers, labels = find_clusters(X, 3, rseed=0)
print("n_clusters: 3, rseed: 0")
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.show()

labels = KMeans(3, random_state=0).fit_predict(X)
print("n_clusters: 3, rseed: 0")
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.show()
```

Результат виконання програми:


4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

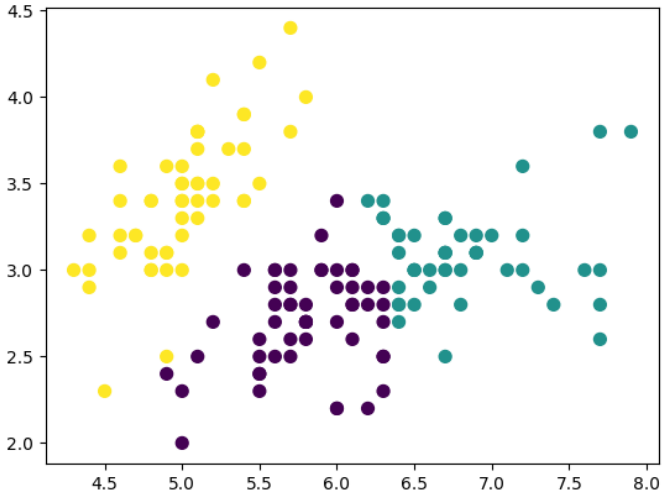

4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

Рис. 8.1 – 8.3. Результат виконання програми

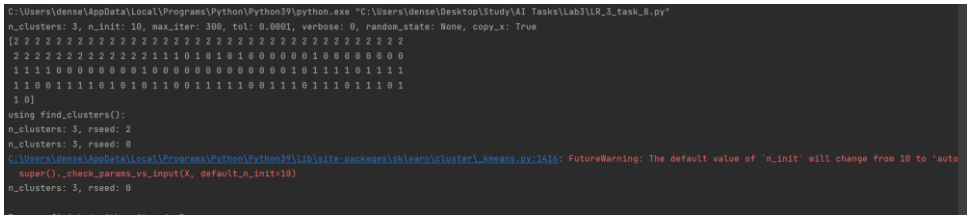


Рис. 8.4 Результат виконання програми

Висновки до завдання

Отже, метод k-середніх може бути використаний для кластеризації даних. Він дозволяє розділити дані на групи, для різної кількості кластерів. Метод k-середніх є дуже швидким методом, що дозволяє обробляти великі обсяги даних. Однак, важливо пам'ятати, що він має свої обмеження. Наприклад, він чутливий до вибору початкових центрів, може не давати оптимальні результати при неправильному виборі кількості кластерів та може неправильно працювати з даними, які не мають чіткого розділення на групи.

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

Відповідно до рекомендацій, напишіть програму та оцініть максимальну кількість кластерів у заданому наборі даних за допомогою алгоритму зсуву середньою. Для аналізу використовуйте дані, які містяться у файлі data_clustering.txt.

Лістинг програми:

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

# Завантаження даних
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, col-
or=np.random.rand(3,))

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

plt.title('Кластери')
plt.show()
```

Результат виконання програми:

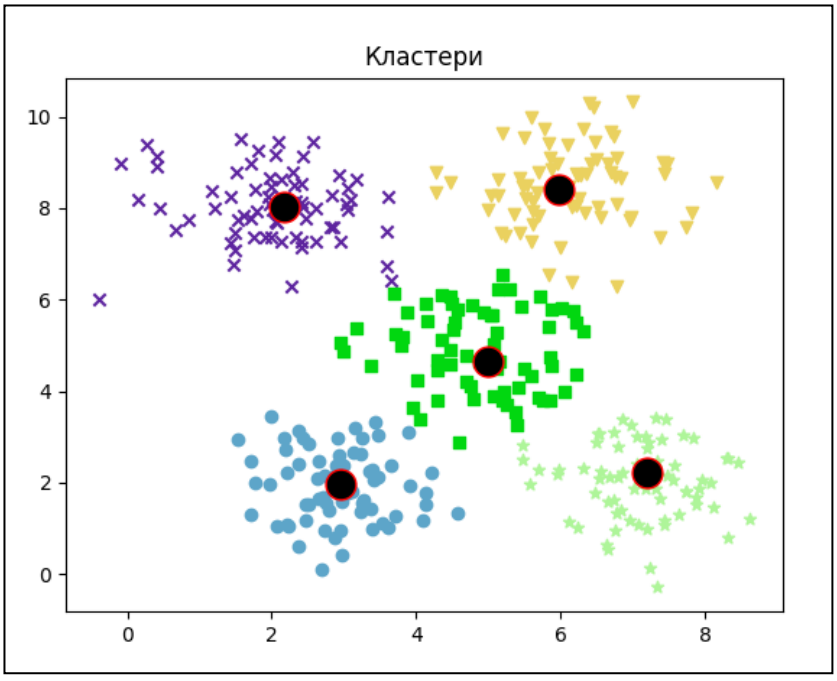


Рис. 9.1. Кластери, які отримані методом зсуву середнього

```
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. 9.2. Центри кластерів

Висновки до завдання

Метод зсуву середнього є потужним алгоритмом, оскільки він не вимагає жодних припущень про базовий розподіл даних і може працювати з різноманітними просторами функцій. Важливу роль відіграє обрана ширина вікна (bandwidth). Як можна помітити, кількість кластерів у цьому методі співпадає з результатами попереднього завдання.

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

Використовуючи модель поширення подібності, знайти підгрупи серед учасників фондового ринку. У якості керуючих ознак будемо використовувати варіацію котирувань між відкриттям і закриттям біржі.

Лістинг програми:

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

input_file = 'company_symbol_mapping.json'
with open(input_file, 'r') as f:
    company_symbols_map = json.loads(f.read())

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

start_date = datetime.datetime(2003, 7, 3)
end_date = datetime.datetime(2007, 5, 4)

quotes = []
for symbol in symbols:
```

```
    quote = yf.Ticker(symbol).history(start=start_date, end=end_date)
    quotes.append(quote)

opening_quotes = np.array([quote['Open'].values for quote in
quotes]).astype(float)
closing_quotes = np.array([quote['Close'].values for quote in
quotes]).astype(float)

quotes_diff = closing_quotes - opening_quotes

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

edge_model = covariance.GraphicalLassoCV()

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

_, labels = cluster.affinity_propagation(edge_model.covariance_)
num_labels = labels.max()

for i in range(num_labels + 1):
    cluster_names = names[labels == i]
    if len(cluster_names) > 0:
        print("Cluster", i+1, "==>", ', '.join(cluster_names))
```

Результат виконання програми:

```
Cluster 1 ==> Microsoft, IBM, Amazon, Ford, 3M, Mc Donalds, Apple
Cluster 2 ==> Northrop Grumman, Boeing
Cluster 3 ==> Coca Cola, Pepsi, Kellogg, Procter Gamble
```

Рис. 10. Результат виконання програми

Репозиторій: https://github.com/SerhiiDenysiuk23/AI_Labs

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