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*.

ЧАСТИНА 1. ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Теоретичні відомості подані на лекціях. Також доцільно вивчити матеріал поданий в літературі:

Джоши Пратик. Искусственный интеллект с примерами на Python. : Пер. с англ. - СПб. : ООО "Диалектика", 2019. - 448 с. - Парал. тит. англ. ISBN 978-5-907114-41-8 (рус.)

Можна використовувати Google Colab або Jupiter Notebook.

Регресія - це процес оцінки того, як співвідносяться між собою вхідні та вихідні змінні. Слід зазначити, що вихідні змінні можуть мати значення з безперервного ряду дійсних чисел. Отже, існує безліч результуючих можливостей. Це різко контрастує з процесом класифікації, у якому кількість вихідних класів фіксовано.

У регресії передбачається, що вихідні змінні залежить від вхідних, і завдання полягає у з'ясуванні співвідношення між ними. Звідси вхідні змінні називаються незалежними змінними (або предикторами), а вихідні – залежними (або критеріальними змінними). При цьому не потрібно, щоб вхідні змінні були незалежними один від одного. Існує безліч ситуацій, коли між вхідними змінними існує кореляція.

Регресійний аналіз дозволяє з'ясувати, як змінюється значення вихідний змінної, коли змінюємо лише частина вхідних змінних, залишаючи інші вхідні змінні фіксованими. У разі лінійної регресії передбачається, що вхідні та вихідні змінні пов'язані між собою лінійною залежністю. Це накладає обмеження на нашу процедуру моделювання, але прискорює її та робить більш ефективною.

Іноді лінійної регресії виявляється недостатньо для пояснення співвідношень між вхідними та вихідними змінними. У подібних випадках ми використовуємо поліноміальну регресію, в якій вхідні та вихідні змінні пов'язані між собою поліноміальною залежністю.

З обчислювальної точки зору такий підхід складніший, але забезпечує більш високу точність. Вибір виду регресії для виявлення зазначених відношень визначається видом конкретної задачі. Регресію часто використовують для прогнозування цін, економічних показників та інше.

2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ

Завдання 2.1. Створення регресора однієї змінної

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

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

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

Завантажуємо вхідні дані.

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

У цьому текстовому файлі використовується кома, тому для завантаження даних можна скористатися наступним викликом функції.

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

Розіб'ємо дані на навчальний та тестовий набори.

```
# 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:]
```

Створимо об'єкт лінійного регресора та навчимо його, використовуючи тренувальні дані.

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

regressor.fit(X_train, y_train)
```

Спрогнозуємо результат для тестового набору даних, використовуючи модель, що навчається.

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

Побудуємо вихідний графік.

```
# 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()
```

Обчислимо метричні параметри регресора, порівнюючи справжні значення з передбаченими.

```
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))
```

Створивши модель, ми можемо зберегти її у файлі для подальшого використання. Python надає відмінний модуль, який дозволяє легко це зробити.

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

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

Завантажимо модель з файлу на диску та побудуємо прогноз.

```
# 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))
```

Збережіть код робочої програми під назвою LR_3_task_1.py
Код програми, графік функції та результати оцінки якості занесіть у звіт.
Зробіть висновок

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

№ за списком	11	12	13	14	15	16	17	18	19	20
№ варіанту	1	2	3	4	5	1	2	3	4	5

№ за списком	21	22	23	24	25	26	27	28	29	30
№ варіанту	1	2	3	4	5	1	2	3	4	5

Варіант 1 файл: data_regr_1.txt
Варіант 2 файл: data_regr_2.txt
Варіант 3 файл: data_regr_3.txt
Варіант 4 файл: data_regr_4.txt
Варіант 5 файл: data_regr_5.txt

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Зробити по аналогії з пунктом 2.1.

Збережіть код робочої програми під назвою LR_3_task_2.py
Код програми, графік функції та результати оцінки якості занесіть у звіт.
Зробіть висновок

Завдання 2.3. Створення багатовимірного регресора

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

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

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

Відкрийте файл, , який містить дані: data_multivar_regr.txt.
У цьому текстовому файлі в якості роздільника використовується кома.
Завантажте данні .
Розбийте дані на навчальний та тестовий набори (як в завданні 2.1).
Створіть та навчіть модель **лінійного регресора** (як в завданні 2.1).
Спрогнозуйте результат для тестового набору даних.

Виведіть на екран метрики якості лінійної регресії.
Linear Regressor performance: Mean absolute error, Mean squared error, Median absolute error, Explained variance score, R2 score.

Створіть **поліноміальний регресор** ступеня 10 та навчіть його на тренувальних даних.

```
# Поліноміальна регресія
polynomial = PolynomialFeatures(degree=10)
X_train_transformed = polynomial.fit_transform(X_train)
```

Візьміть деяку вибіркову точку даних і спрогнозуйте для неї результат. Перший крок полягає в тому, щоб перетворити її на поліном.

```
datapoint = [[7.75, 6.35, 5.56]]
poly_datapoint = polynomial.fit_transform(datapoint)
```

Неважко зазначити, що ця точка дуже близька до точки даних [7.66, 6.29, 5.66], зазначеної в рядку 11 нашого файлу даних. Тому вдало створений регресор повинен передбачити результат, близький до 41.35. Створіть об'єкт лінійного регресора і виконайте підгонку до полінома. Побудуйте прогноз з використанням як лінійного, так і поліноміального регресора, щоб побачити різницю.

```
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))
```

Оцініть та порівняйте отримані характеристики.

Зверніть увагу, що порівняно з лінійним регресором поліноміальний регресор забезпечує отримання результату, ближчого до значення 41.35. Тобто дає кращі результати.

Збережіть код робочої програми під назвою LR_3_task_3.py
Код програми та результати оцінки якості занесіть у звіт.
Зробіть висновок

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). Оригінальні дані можна завантажити з: https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html'.

Використайте всі функції набору даних про діабет, щоб побудувати двовимірний графік лінійної регресії. Побудуйте графік залежності між спостережуваними відповідями в наборі даних і відповідями, передбаченими лінійним наближенням (крапками) та пряму лінію, по цьому графіку, що покаже, як лінійна регресія намагається провести пряму лінію, яка мінімізує залишкову суму квадратів між спостережуваними відповідями в наборі даних і відповідями, передбаченими лінійним наближенням. Також розрахуйте коефіцієнт кореляції R2, середню абсолютну помилку (MAE) і середньоквадратичну помилку (MSE).

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Вам знадобляться:

```
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
```

Завантажте дані

```
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target
```

Поділіть їх на навчальну та тестову вибірки з параметрами test_size = 0.5, random_state = 0

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size
= 0.5, random_state = 0)
```

Створіть модель лінійної регресії та натренуйте її.

```
regr = linear_model.LinearRegression()
```

```
regr.fit(Xtrain, ytrain)
```

Зробіть прогноз по тестовій вибірці

```
ypred = regr.predict(Xtest)
```

Розрахуйте, підпишіть та виведіть на екран коефіцієнти регресії та показники

regr.coef_
regr.intercept_
r2_score
mean_absolute_error
mean_squared_error

Побудуйте графіки

```
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()
```

Збережіть код робочої програми під назвою LR_3_task_4.py
Код програми, графіки та результати оцінки якості занесіть у звіт.
Зробіть висновок

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

№ за списком	11	12	13	14	15	16	17	18	19	20
№ варіанту	1	2	3	4	5	6	7	8	9	10

№ за списком	21	22	23	24	25	26	27	28	29	30
№ варіанту	1	2	3	4	5	6	7	8	9	10

Варіант 1

```
m = 100
X = 6 * np.random.rand(m, 1) - 5
y = 0.5 * X ** 2 + X + 2 + np.random.randn(m, 1)
```

Варіант 2

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

Варіант 3

```
m = 100
X = 6 * np.random.rand(m, 1) - 4
y = 0.5 * X ** 2 + X + 2 + np.random.randn(m, 1)
```

Варіант 4

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

Варіант 5

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

Варіант 6

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

Варіант 7

```
m = 100
X = np.linspace(-3, 3, m)
y = np.sin(X) + np.random.uniform(-0.5, 0.5, m)
```

Варіант 8

```
m = 100
X = np.linspace(-3, 3, m)
y = 2 * np.sin(X) + np.random.uniform(-0.5, 0.5, m)
```

Варіант 9

```
m = 100
X = np.linspace(-3, 3, m)
y = 3 + np.sin(X) + np.random.uniform(-0.5, 0.5, m)
```

Варіант 10

```
m = 100
X = np.linspace(-3, 3, m)
y = 4 + np.sin(X) + np.random.uniform(-0.6, 0.6, m)
```

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Побудуйте вашу модель, та виведіть ваші дані на графік залежності *y=f(X)*.

Що, якщо ваші дані насправді складніші за звичайну пряму лінію? Дивно, але ви насправді можете застосовувати лінійну модель для припасування до нелінійних даних. Простий спосіб передбачає додавання ступенів кожної ознаки у вигляді нових ознак і наступне навчання лінійної моделі на такому розширеному наборі ознак. Цей прийом називається *поліноміальною регресією (polynomial regression)*.

Якщо вхідні данні розподілені нелінійно, то, безумовно, пряму лінію ніколи не буде підігнано під такі дані належним чином. Тому скористайтеся класом PolynomialFeatures з Scikit-Learn, щоб перетворити наші навчальні дані, додавши як нові ознаки квадрат (поліном 2-го ступеня) кожної ознаки (у нашому випадку є тільки одна ознака):

```
PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
```

Виведіть значення коефіцієнтів полінома X[0] та X_poly на екран.

Тепер X_poly містить первинну ознаку X плюс її квадрат.

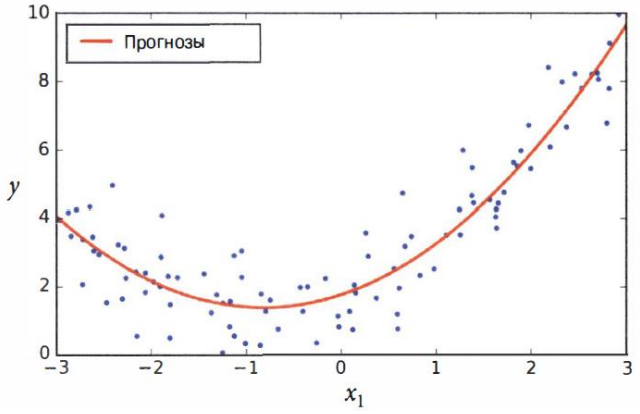
Далі підгоніть модель LinearRegresion до таких розширених навчальних даних.

```
lin_reg = LinearRegression( )
lin_reg.fit(X_poly, y)
lin_reg.intercept_, lin_reg.coef_
```

Зверніть увагу, що за наявності множини ознак поліноміальна регресія здатна знайти зв'язок між ознаками (те, що проста лінійна регресійна модель робити неспроможна). Це стає можливим завдяки тому факту, що клас PolynornialFeatures також додає всі комбінації ознак до заданого ступеня.

Наприклад, якщо є дві ознаки a і b, тоді PolynornialFeatures з degree=3 додав би як ознаки a2, a3, b2 і b3, а й комбінації ab, a2b і ab2.

Виведіть графік вашої моделі крапками, а регресію лінією. Наприклад:



Ваш графік занесіть у звіт.

Запишіть модель вашого варіанта у вигляді математичного рівняння (наприклад $y = 0.5x_1^2 + 1.0x_1 + 2.0 + гауссов\ шум$***) та запишіть отриману вами модель регресії з передбаченими коефіцієнтами (наприклад*** $y = 0.56x_1^2 + 0.93x_1 + 1.78$***).***

Отримані вами коефіцієнти повинні бути близьким до модельних. І це буде означати що модель навчена правильно.

Збережіть код робочої програми під назвою LR_3_task_5.py
Код програми та результати регресії занесіть у звіт.
Зробіть висновок

Завдання 2.6. Побудова кривих навчання

Побудуйте криві навчання для ваших даних у попередньому завданні.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

У разі виконання поліноміальної регресії високого ступеня, ймовірно, ви краще підганяєте навчальні дані, ніж за допомогою лінійної регресії. Наприклад, на рис. 4 демонструється застосування поліноміальної моделі 300-го ступеня до попередніх навчальних даних, а результат порівнюється з чистою лінійною моделлю та квадратичною моделлю (поліноміальною 2-го ступеня).

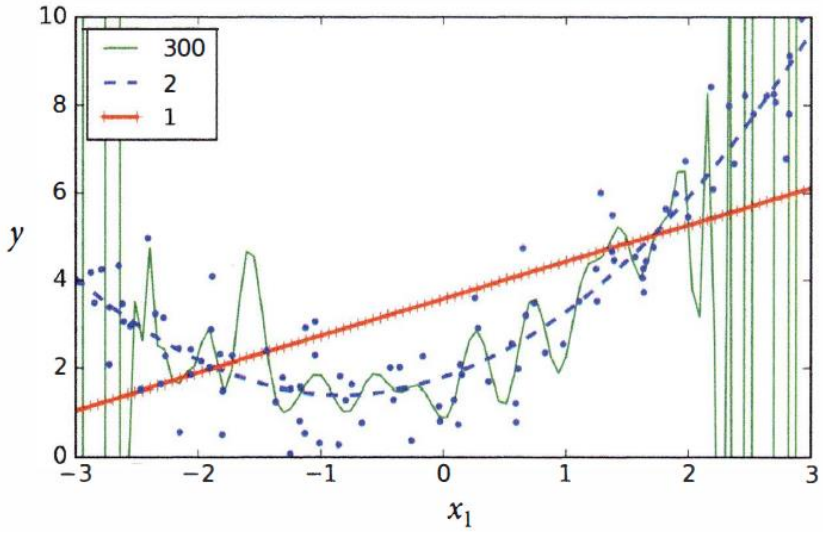


Рис.4

Зверніть увагу, як поліноміальна модель 300-го ступеня коливається, щоб якомога більше наблизитися до навчальних зразків. Зрозуміло, така поліноміальна регресійна модель високого ступеня викликає сильне перенавчання навчальними даними, тоді як лінійна модель - недонавчання ними. У даному випадку добре узагальнюватиметься квадратична модель. Це має сенс, оскільки дані генерувалися з використанням квадратного рівняння, але з урахуванням того, що зазвичай ви *не знатимете функцію*, що використовується для генерації даних, як приймати рішення про те, наскільки складною має бути модель? Як з'ясувати, що модель перенавчається або недонавчається на даних?

У попередніх лабораторних роботах за допомогою перехресної перевірки оцінювалася продуктивність узагальнення моделі. Якщо згідно з метриками перехресної перевірки модель добре виконується на навчальних даних, але погано узагальнюється, то модель перенавчена. Якщо модель погано виконується в обох випадках, тоді вона недонавчена. Так виглядає один із способів сказати, що модель надто проста чи надмірно складна.

Інший спосіб передбачає перегляд кривих навчання (learning curve): вони є графіки продуктивності моделі на навчальному наборі і перевірочному наборі як функції від розміру навчального набору (або ітерації навчання). Щоб отримати такі графіки, потрібно просто навчити модель кілька разів на підмножині різних розмірів, взятих з навчального набору.

Визначте функцію, яка будує криві навчання моделі для встановлених навчальних даних:

```
from sklearn.metrics import mean_squared_error
from sklearn model_selection import train_test_split

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")
```

Побудуйте криві навчання для звичайної лінійної регресійної моделі із попереднього завдання (ваш варіант):

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

Ви повинні отримати щось подібне

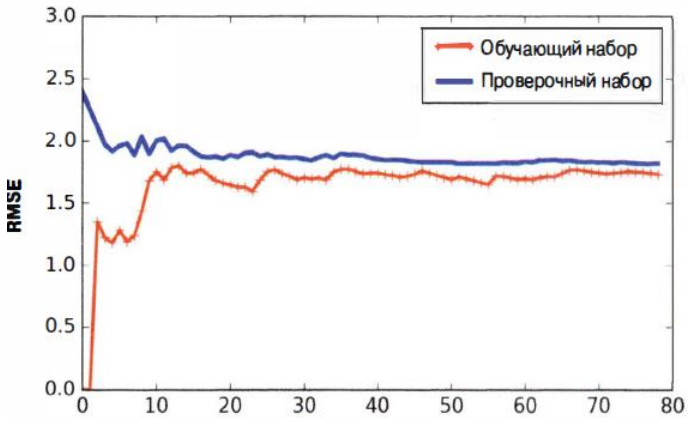


Рис. 5.Криві навчання для лінійної моделі

Ваш графік занесіть у звіт.

Тут потрібні деякі пояснення. Насамперед зверніть увагу на продуктивність моделі у разі використання навчальних даних: коли в навчальному наборі є тільки один або два зразки, модель може бути повною мірою підігнана до них, що пояснює початок кривої з нульової помилки. Але в міру додавання зразків у навчальний набір ідеальна підгонка моделі до навчальних даних стає неможливою, як через те, що дані зашумлені, так і тому, що вони зовсім відрізняються від лінійних. Отже, помилка на навчальних даних рухається вгору, поки не стабілізується, коли додавання

нових зразків у навчальний набір не робить середню помилку набагато краще чи гірше. Тепер перейдемо до продуктивності моделі на перевірочних даних. Коли модель навчалася на незначній кількості зразків, вона нездатна узагальнюватися належним чином, а тому помилка перевірки спочатку досить велика. Потім у міру того, як модель бачить все більше навчальних зразків вона навчається, а помилка перевірки відповідно повільно знижується. Однак пряма лінія знову не в змозі добре змоделювати дані, тому помилка стабілізується поблизу іншої кривої.

Такі криві навчання типові для недонавченої моделі. Обидві криві стабілізуються; вони розташовані близько одна до одної і знаходяться досить високо.

Якщо ваша модель недонавчена на навчальних даних, тоді додавання додаткових навчальних зразків не допоможе. Вам потрібно вибрати складнішу модель або знайти найкращі ознаки.

Тепер побудуйте криві навчання поліноміальної моделі 10-го ступеня на тих самих ваших даних:

```
from sklearn.pipeline import Pipeline

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

plot_learning_curves(polynomial_regression, X, y)
```

Ви повинні отримати щось подібне до рис. 6.

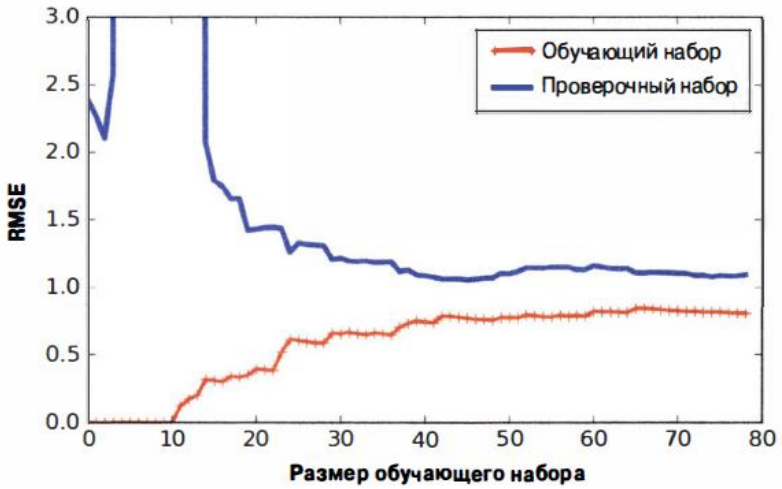


Рис.6. Криві навчання для поліноміальної моделі

Ваш графік занесіть у звіт.

Криві навчання виглядають трохи краще за попередні, але є дві дуже важливі відмінності.

- Помилка на навчальних даних набагато нижча, ніж у випадку лінійної регресійної моделі.

- Між кривими є проміжок. Це означає, що модель виконується значно краще на навчальних даних, ніж на перевірочних даних, демонструючи ознаку перенавчання. Тим не менш, якщо ви застосуєте набагато більший навчальний набір, дві криві продовжать зближення.

Один із способів поліпшення перенавченої моделі полягає у наданні їй додаткових навчальних даних доти, доки помилка перевірки не досягне помилки навчання.

Компроміс між зміщенням та дисперсією

Важливим теоретичним результатом статистики та машинного навчання є той факт, що помилка узагальнення моделі може бути виражена у вигляді суми трьох різних помилок.

Зміщення. Ця частина помилки узагальнення пов'язана з невірними припущеннями, такими як припущення того, що дані є лінійними, коли вони насправді квадратичні. Модель з високим зсувом, швидше за все, недонавчиться на навчальних даних.

Дисперсія. Ця частина пояснюється надмірною чутливістю моделі до невеликих змін у навчальних даних. Модель з багатьма ступенями свободи (така як поліноміальна модель високого ступеня), ймовірно, матиме високу дисперсію і тому перевчитися навчальними даними.

Непереборна похибка. Ця частина з'являється внаслідок шуму самих даних. Єдиний спосіб скоротити непереборну похибку в помилці передбачає очищення даних (наприклад, упорядкування джерел даних, таких як несправні датчики, або виявлення та усунення викидів).

Зростання складності моделі зазвичай збільшує її дисперсію та зменшує зміщення. І навпаки, скорочення складності моделі збільшує її зміщення та зменшує дисперсію. Ось чому це називається компромісом.

Тепер побудуйте криві навчання поліноміальної моделі 2-го ступеня на тих самих ваших даних.

Ваш отриманий графік занесіть у звіт.

Код програми та результати занесіть у звіт.

Програмний код збережіть під назвою LR_3_task_6.py

ЧАСТИНА 2. ДОСЛІДЖЕННЯ МЕТОДІВ НЕКОНТРОЬОВАНОГО НАВЧАННЯ

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Термін *навчання без вчителя (unsupervised learning)* відноситься до процесу побудови моделі машинного навчання, що не вимагає залучення розмічених тренувальних даних. Машинне навчання без вчителя знаходить застосування у багатьох галузях, включаючи сегментування ринку, торгівля акціями, обробка природної мови, машинний зір та ін.

У попередніх ЛР ми мали справу з даними, з якими асоціювалися позначки (маркери). У разі помічених навчальних даних алгоритми вчаться класифікувати по цих мітках.

Алгоритми навчання без вчителя намагаються будувати моделі, які здатні знаходити підгрупи в заданому наборі даних, використовуючи різні метрики подібності.

Розглянемо, як формулюється завдання навчання, якщо воно проводиться без учителя. Коли у нас є набір даних, які не асоціюються з будь-якими мітками, ми припускаємо, що ці дані генеруються під впливом прихованих змінних, що управляють їх розподілом. У такому разі процес навчання може наслідувати якусь ієрархічну схему, використовуючи на початковому етапі індивідуальні точки даних. Далі можна створювати більш глибокі рівні представлення даних.

2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ

Завдання 2.7. Кластеризація даних за допомогою методу k-середніх

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

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Кластеризація - один із найпопулярніших методів навчання без вчителя. Ця методика застосовується для аналізу даних та виділення кластерів серед них. Для знаходження кластерів застосовують різні метрики подібності, такі як евклідова відстань, що дозволяють виділяти підгрупи даних. Використовуючи міру подібності, можна оцінити складність кластера. Таким чином, кластеризація - це процес організації даних у підгрупи, елементи яких подібні між собою відповідно до деяких критеріїв.

Наше завдання полягає в тому, щоб ідентифікувати приховані властивості точок даних, що визначають їхню приналежність до однієї і тієї ж підгрупи. Універсальних метричних параметрів подібності, які б у всіх випадках працювали, не існує. Все визначається конкретикою завдання. Наприклад, нас може цікавити знаходження представницької точки даних для кожної підгрупи або викидів. Залежно від ситуації ми вибираємо ту або іншу метрику, яка, на нашу думку, найбільш повно навчає специфіку завдання.

Метод k-середніх (k-means) - це добре відомий алгоритм кластеризації. Його використання передбачає, що кількість кластерів заздалегідь відома. Далі ми сегментуємо дані до підгруп, застосовуючи різні атрибути даних. Ми починаємо з того, що фіксуємо кількість кластерів та, виходячи з цього, класифікуємо дані. Основна ідея полягає в оновленні положень центроїдів (центрів тяжіння кластеру, або головні точки) на кожній ітерації. Ітеративний процес продовжується до тих пір, поки всі центроїди не займуть оптимального положення.

Як неважко здогадатися, у цьому алгоритмі вибір початкового розташування центроїдів відіграє дуже важливу роль, оскільки це безпосередньо впливає на кінцеві результати. Одна із стратегій полягає в тому, щоб центроїди розташовувалися на якомога більшій відстані один від одного. Базовому методу k-середніх відповідає випадкове розташування центроїдів, тоді як у вдосконаленому варіанті методу (k-means++) ці точки вибираються алгоритмічно з списку вхідних точок даних. На початку процесу робиться спроба розташувати центри кластерів на великих відстанях один від одного, щоб забезпечити швидку збіжність. Потім ми перебираємо дані навчального набору та покращуємо стартове розбиття на кластери за допомогою віднесення кожної точки до найближчого кластерного центру.

Завершення перебору всіх точок набору даних означає закінчення першої ітерації. У цьому етапі точки виявляються згрупованими виходячи з початкових положень центрів кластерів. Далі нам необхідно заново вирахувати положення центроїдів, відштовхуючись від нових кластерів, отриманих наприкінці першої ітерації. Отримавши новий набір до центрів, ми повторюємо весь процес, знову ітеруючи по набору даних і відносячи кожну точку до найближчого центроїду.

У процесі повторення описаних кроків центри кластерів поступово зміщуються до своїх стійких положень. Після виконання певної кількості ітерацій центри кластерів перестануть зміщуватися. Це свідчить про те, що ми досягли сталого розташування центрів кластерів. Отримані K центроїдів і являють собою остаточну модель k-середніх, які будуть використовуватися для виведення суджень (inference).

Щоб подивитися, як працює метод кластеризації k-середніх, застосуємо його до двовимірних даних. Будемо використовувати дані, що містяться у файлі data_clustering txt. У цьому файлі кожен рядок містить два числа, розділені комою.

У scikit-learn K-means реалізується як об’єкт кластера, який називається sklearn.cluster.KMeans, і використовується для пошуку кластерів.

Створіть новий файл Python та імпортуйте такі пакети.

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

Завантажимо вхідні дані із файлу.

```
# Завантаження вхідних даних
X = np.loadtxt('data_clustering.txt', delimiter=',')
```

Щоб застосувати k-середніх необхідно задати кількість кластерів

```
num_clusters = 5
```

Візуалізуйте вхідні дані, щоб побачити, як виглядає розподіл.

```
# 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('Входные данные')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
```

Графік занесіть у звіт.

Ми можемо отримати наочне підтвердження, що наші дані складаються з п'яти груп. Створимо об'єкт KMeans, використовуючи параметри ініціалізації.

Параметр init дозволяє встановити спосіб ініціалізації початкових центрів кластерів. Замість того, щоб вибирати їх випадковим чином, ми використовуємо для цього параметра значення k-means++, яке забезпечує покращений спосіб вибору положень центроїдів, що гарантує швидку збіжність алгоритму. Параметр n_clusters визначає кількість кластерів, тоді як параметр n_init дозволяє вказати, скільки разів повинен виконатися алгоритм, перш ніж буде прийнято рішення щодо найкращого результату.

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

Навчимо модель k-середніх на вхідних даних.

```
# Навчання моделі кластеризації KMeans
kmeans.fit(X)
```

Щоб візуалізувати межі, ми маємо створити сітку точок та обчислити модель на всіх вузлах сітки. Визначимо крок сітки.

```
# Визначення кроку сітки
step_size = 0.01
```

Далі визначимо саму сітку і переконаємось у тому, що вона охоплює всі вхідні значення.

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

Спрогнозуйте результати всіх точок сітки, використовуючи навчену модель k-середніх.

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

Відобразіть на графіку вихідні значення та виділіть кожну область своїм кольором.

```
# 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')
```

Відобразіть вхідні дані на виділених кольором областях.

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

Відобразіть на графіку центри кластерів, отримані з використанням методу k-середніх.

```
# Відображення центрів кластерів
```

```
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('Границы кластеров')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

Ваш графік занесіть у звіт.

Збережіть код робочої програми під назвою LR_3_task_7.py
Код програми, графік функції та результати оцінки якості занесіть у звіт.
Зробіть висновок

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.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Можна написати свій власний код програми з поясненнями по аналогії з попереднім завданням або скористатися підказками, але у цьому випадку ви повинні прокоментувати кожен рядок чи функцію коду де є позначка #.

Код підказки (містить помилки):

```
sklearn.svm import SVC
from sklearn.metrics import pairwise_distances_argmin

import numpy as np
iris = load_iris()
X = iris['data']
```

```
y = iris['target']
#
sklearn.cluster.KMeans(n_clusters = 8, init = 'k-means + +', n_init = 10, max_iter =
300, tol = 0.0001, precompute_distances = 'auto', verbose = 0, random_state =
None, copy_x = True, n_jobs = None, algorithm = 'auto')
#
kmeans = KMeans(n_clusters = 5)
#
kmeans.fit(X)
#
y_kmeans = kmeans.predict(X);
#
plt.scatter(X[:, 0], X[:, 1], c = y_kmeans, s = 50, cmap = 'viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c = 'black', s = 200, alpha = 0.5);

#
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
centers, labels = find_clusters(X, 3)
plt.scatter(X[:, 0], X[:, 1], c = labels,
s = 50, cmap = 'viridis');
#
centers, labels = find_clusters(X, 3, rseed = 0)
plt.scatter(X[:, 0], X[:, 1], c = labels,
s = 50, cmap = 'viridis');
#
```

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

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_3_task_8.py

Код програми та рисунок занесіть у звіт.

Зробіть висновок

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.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Метод зсуву середнього (Mean Shift) - потужний алгоритм, що використовується в навчанні без вчителя. Цей непараметричний алгоритм часто застосовується при вирішенні завдань кластеризації. Він називається не параметричним, оскільки в ньому не використовуються будь-які припущення щодо базового розподілу даних.

Цей метод контрастує з параметричними підходами, у яких передбачається, що базові дані підпорядковуються стандартному розподілу ймовірностей. Метод зсуву середнього знаходить безліч застосувань у таких областях, як відстеження об'єктів та аналіз даних у реальному часі.

У алгоритмі зсуву середнього весь простір ознак сприймається як функція розподілу ймовірності. Ми починаємо з тренувального набору даних і припускаємо, що ця вибірка відповідає функції розподілу ймовірності. В рамках такого підходу кластери відповідають максимуму базового розподілу. Якщо існують *K* кластерів, то в базовому розподілі існують *K* піків, і метод зсуву середнього ідентифікує ці вершини.

Метою методу зсуву середнього є ідентифікація позицій центрів кластерів. Для кожної точки навчального набору визначається оточуюче її вікно. Потім для цього вікна визначається центроїд і положення вікна оновлюється так, щоб воно відповідало стану нового центроїду. Далі процес повторюється для нового центроїду шляхом визначення вікна навколо нього. У міру продовження описаного процесу ми наближаємось до піку кластера. Кожна точка даних переміщатиметься у напрямку кластера, якому вона належить. Це переміщення здійснюється у напрямку області з більш високою щільністю ймовірності.

Ми продовжуємо процес зміщення центроїдів, що також звуться середніми, до піків кожного кластера. Оскільки середні при цьому зміщуються, метод і називається *зсув середнього*. Цей процес триває до того часу, поки алгоритм не зійдеться, тобто. поки центроїди не перестануть зміщуватися.

Створіть новий файл Python та імпортуйте такі пакети.

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

Завантажимо вхідні дані.

```
# Завантаження
X = np.loadtxt('data_clustering.txt', delimiter=',')
```

Зверніть увагу на ширину вікна вхідних даних. *Ширина вікна* (bandwidth) - це параметр базового процесу оцінки щільності розподілу ядра в алгоритмі зсуву середньою. Ширина вікна впливає на загальну швидкість збіжності алгоритму та результуючу кількість кластерів. Отже, цей параметр відіграє важливу роль. Вибір занадто малої ширини вікна може призвести до занадто великої кількості кластерів, тоді як завищені значення цього параметра призводять до злиття окремих кластерів.

Параметр quantile впливає на ширину вікна. Вищі значення цього параметра збільшують ширину вікна, тим самим зменшуючи кількість кластерів.

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

Навчимо модель кластеризації на основі зсуву середнього, використовуючи отриману оцінку ширини вікна.

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

Витягнемо центри всіх кластерів.

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

Витягнемо кількість кластерів.

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

Візуалізуємо точки даних.

```
    # Відображення на графіку точок та центрів кластерів
plt.figure()
markers = 'o*xvs'
for i, marker in zip(range(num_clusters), markers):
    # Отображение на графике точек, принадлежащих
    # текущему кластеру
    plt.scatter(X[labels==i, 0], X[labels==i, 1], marker=marker,
                color='black')
```

Відобразимо на графіку центр поточного кластера.

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

Після виконання цього коду на екрані відобразиться графік. У вікні термінала відобразяться координати центрів кластерів.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_3_task_9.py

Код програми та рисунок занесіть у звіт.

Зробіть висновок

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.

Використовувати файл вхідних даних фондового ринку, що доступний в бібліотеці matplotlib. Прив'язки символічних позначень компаній до повних назв містяться у файлі company_symbol_mapping.json.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Поширення подібності (affinity propagation) - це алгоритм кластеризації, виконання якого вимагає попередньої вказівки використовуваної кількості кластерів . У силу своєї спільності та простоти реалізації він широко застосовується у різних галузях. Цей алгоритм знаходить представницькі елементи кластерів, що звуться зразки (exemplars), використовуючи техніку "обміну повідомленнями " між точками даних.

Ми починаємо з визначення мір подібності, які має використовувати алгоритм. Спочатку як потенційні зразки розглядаються всі навчальні точки даних. Далі точки даних «спілкуються» між собою до тих пір поки не вдасться визначити представницький набір зразків.

Елементи кластерів попарно обмінюються повідомленнями двох категорій, що містять інформацію про придатність (responsibility) та доступність (availability) елементів для ролі зразків. Повідомлення першої категорії надсилаються елементами кластера потенційним зразкам і вказують на те, наскільки добре точка даних підходила б для того, щоб бути елементом кластера даного зразка. Повідомлення другої категорії надсилаються потенційними зразками потенційним елементам кластера і вказують на те, наскільки добре вони підійшли б для того, щоб служити зразком. Цей процес триває до того часу, поки алгоритм не зійдеться оптимального набору зразків.

Також є параметр preference, що управляє кількістю зразків, які повинні бути знайдені. Якщо ви виберете для нього завищене значення, це призведе до того, що алгоритм знайде занадто велику кількість кластерів. Наслідком заниженого значення цього параметра буде занадто мала кількість кластерів.

Створіть новий файл Python та імпортуйте такі пакети.

```
import datetime
import json
import numpy as np
import matplotlib.pyplot as plt
from sklearn import covariance, cluster
from matplotlib.finance import quotes_historical_yahoo_ochl
as quotes_yahoo

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

Завантажте з файлу масив відповідності символів компаній їх повним назвам.

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
```

Завантажте дані котирувань із бібліотеки matplotlib.

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

Обчисліть різниці між котируваннями при відкритті та закритті біржі.

```
# Вилучення котирувань, що відповідають
# відкриттю та закриттю біржі
opening_quotes = np.array([quote.open for quote in
quotes]).astype(np.float)
closing_quotes = np.array([quote.close for quote in
quotes]).astype(np.float)

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

Нормалізуйте дані.

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

Створіть модель графа.

```
# Створення моделі графа
edge_model = covariance.GraphLassoCV()
```

Навчимо модель.

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

Створіть модель кластеризації на основі поширення подібності, використовуючи щойно навчену крайову модель.

Створення моделі кластеризації на основі поширення подібності

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

Виведіть результати.

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

Код програми та результати занесіть у звіт.

Зверніть увагу що отримані кластери можуть відрізнятися у різних студентів для різних налаштувань.

Програмний код збережіть під назвою LR_3_task_10.py

Коди комітити на GitHab. У кожному звіті повинно бути посилання на GitHab.

Назвіть бланк звіту СШІ-ЛР-3-NNN-XXXXX.doc
де NNN – позначення групи
XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в СШІ-ЛР-3-NNN-XXXXX.pdf