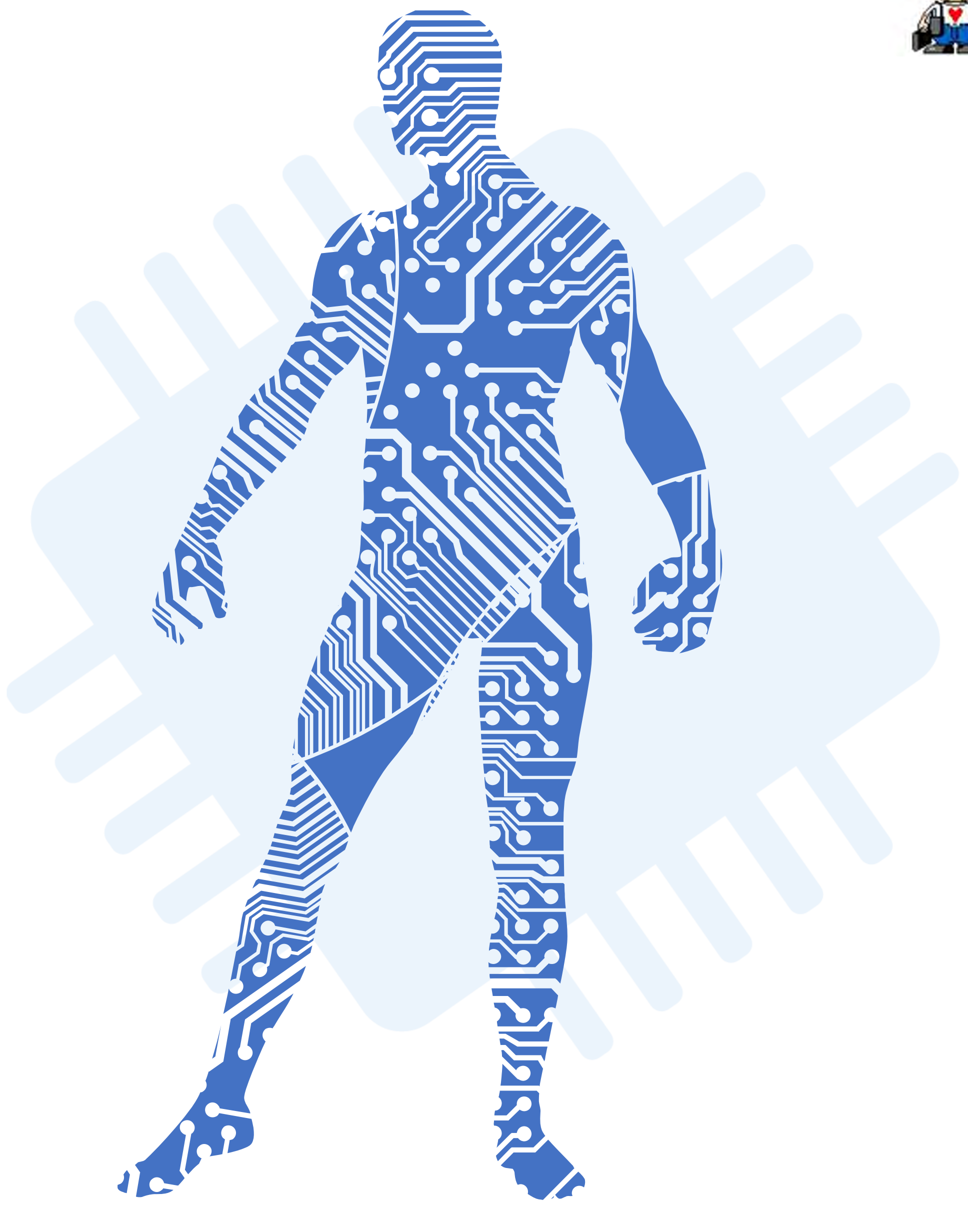
機器學習

第 8 章 多項式迴歸
(Polynomial Regression)

講師：紀俊男

本章大綱

- 多項式迴歸簡介
- 資料前處理
- 多項式迴歸實作
- 使用「快樂版」實作
- 重點整理



多項式迴歸簡介

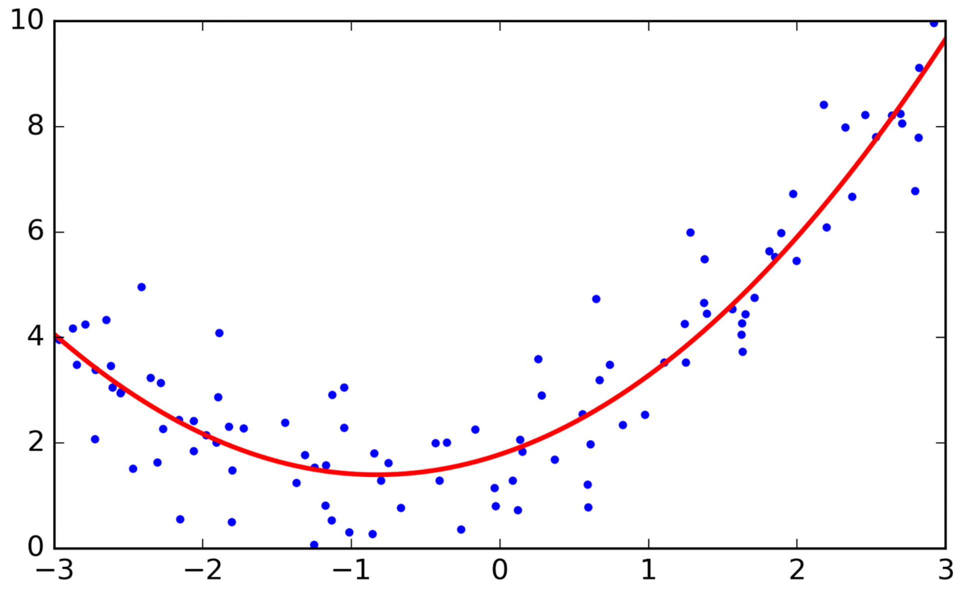
Introduction of Polynomial Regression

何謂「多項式迴歸」

- N 個連續 N 次自變數 vs. 1 個連續應變數（一元 N 次方程式）

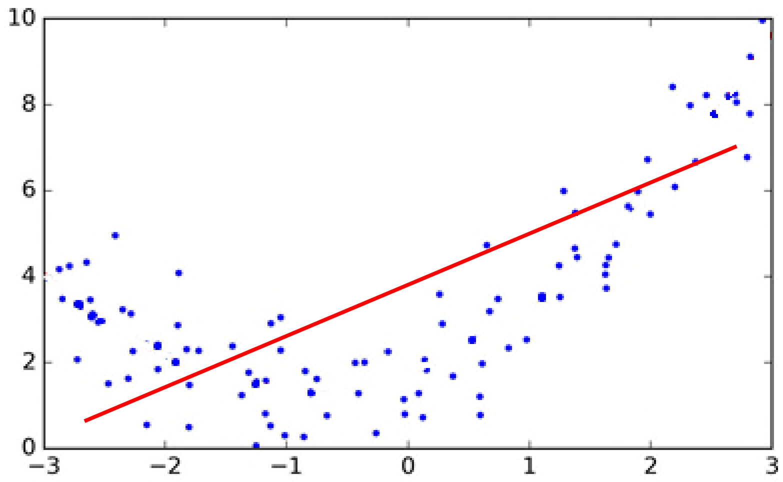
$$f(x) = c_0 + c_1X^1 + \cdots c_nX^n$$

應用場合：
- e.g. 流行病傳播速率

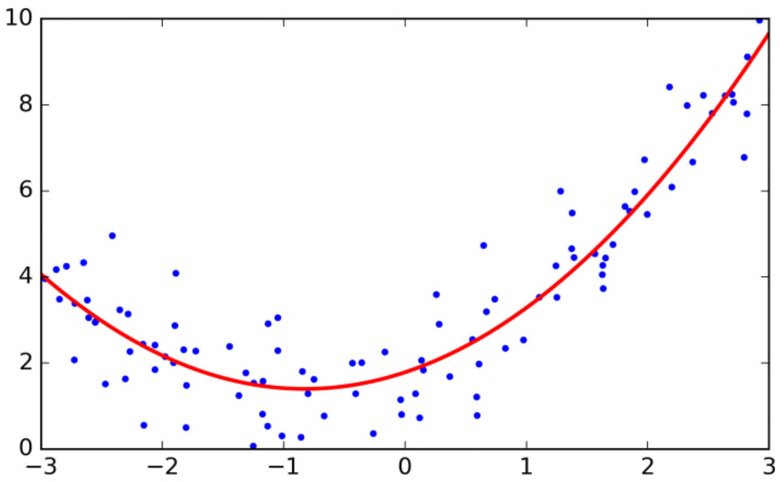


為何使用「多項式迴歸」

- 當**自變數** X_i，對**應變數 Y** 作圖，不是**直線排列**時



不夠精準



比較精準

資料前處理

下載與瀏覽資料集

- 依照講師指示，**下載**並**瀏覽**資料集

Position_Salaries.csv

	A	B	C
1	Position	Level	Salary
2	Business Analyst	1	45000
3	Junior Consultant	2	50000
4	Senior Consultant	3	60000
5	Manager	4	80000
6	Country Manager	5	110000
7	Region Manager	6	150000
8	Partner	7	200000
9	Senior Partner	8	300000
10	C-level	9	500000
11	CEO	10	1000000

目的：利用「職位」--> 推算「薪資」

資料前處理

撰寫程式碼

```python
import HappyML.preprocessor as pp

# Load Dataset
dataset = pp.dataset(file="Position_Salaries.csv")

# Decomposition of Variables
X, Y = pp.decomposition(dataset, x_columns=[1], y_columns=[2])

# Training / Testing Set
X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y, train_size=0.8)
```

資料前處理流程：

1. 載入資料
2. 切分自變數、應變數
3. 處理缺失資料
   （無缺失資料）
4. 類別資料數位化
   （無類別資料）
5. 切分訓練集、測試集
6. 特徵縮放
   （暫無需要）

隨堂練習：資料前處理

- 請先載入下列「自製套件」：
  - import HappyML.preprocessor as pp
- 用下列指令，載入資料集：
  - dataset = pp.dataset(file="Position_Salaries.csv")
- 用下列指令，切分資料集：
  - X, Y = pp.decomposition(dataset, x_columns=[1], y_columns=[2])
  - X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y, train_size=0.8)
- 參考程式碼如下所示：

```python
import HappyML.preprocessor as pp

# Load Dataset
dataset = pp.dataset(file="Position_Salaries.csv")

# Decomposition of Variables
X, Y = pp.decomposition(dataset, x_columns=[1], y_columns=[2])

# Training / Testing Set
X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y, train_size=0.8)
```

多項式迴歸實作

使用「標準函式庫」

以次方倍增加特徵

- 原理

$$f(x) = c_0 + c_1X^1$$

0
1
2
3
4
5
6
7
8
9
10

$$f(x) = c_0 + c_1X^1 + c_2X^2 + c_3X^3 + c_4X^4$$

0	1	2	3	4
1	1	1	1	1
1	2	4	8	16
1	3	9	27	81
1	4	16	64	256
1	5	25	125	625
1	6	36	216	1296
1	7	49	343	2401
1	8	64	512	4096
1	9	81	729	6561
1	10	100	1000	10000

以次方倍增加特徵

- 原始程式碼

```
from sklearn.preprocessing import PolynomialFeatures

poly_feat = PolynomialFeatures(degree=2)
X_poly = poly_feat.fit_transform(X)
```

1. 引入可用「次方倍」增加特徵的類別。
2. 設定要增加到 X^2
3. 將 $c_0+c_1X^1$ 變成 $c_0+c_1X^1+c_2X^2$

X

	0
0	1
1	2
2	3
3	4
4	5

X_poly

	0	1	2
0	1	1	1
1	1	2	4
2	1	3	9
3	1	4	16
4	1	5	25

隨堂練習：以次方倍增加特徵

- 請在上一個練習的程式碼後方，輸入下列原始碼，並予以執行：

```python
# In[] Test for Polynomial Features
from sklearn.preprocessing import PolynomialFeatures
import pandas as pd

# Add the X-squared feature
poly_feat = PolynomialFeatures(degree=2)
X_poly = pd.DataFrame(poly_feat.fit_transform(X))
```

為了保持後續快樂版
程式碼執行正常
改用 DataFrame 格式

- 打開「**變數觀察面板**」，比較 X 與 X_poly 的區別

X

	0
0	1
1	2
2	3
3	4
4	5

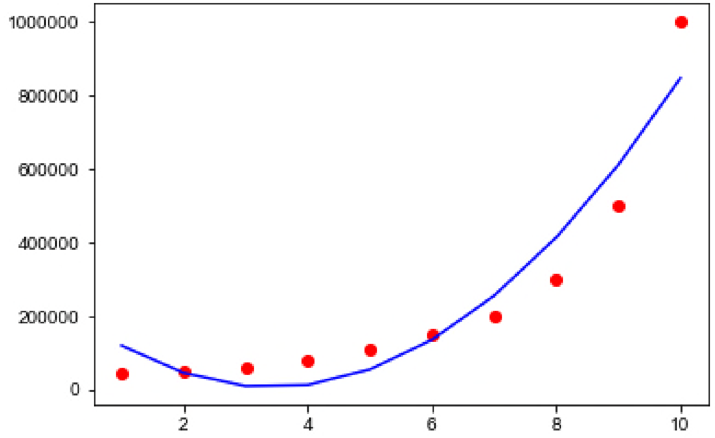
X_poly

	0	1	2
0	1	1	1
1	1	2	4
2	1	3	9
3	1	4	16
4	1	5	25

使用二次方訓練與預測

- 原始程式碼

```
from HappyML.regression import SimpleRegressor
import HappyML.model_drawer as md

model = SimpleRegressor()
model.fit(X_poly, Y)
Y_pred = model.predict(X_poly)

md.sample_model(sample_data=(X, Y), model_data=(X, Y_pred))
```



- 為何使用 SimpleRegressor
  - $C_0+C_1X+C_2X^2+\dots C_nX^n$，可以視為 $C_0+C_1P+C_2Q+\dots C_nR$ 這種多元線性迴歸
  - 那為何不用 MultipleRegressor：也可以！但我們不需要降維，加上兩者都是用「**最小平方法（OLS）**」，殺雞不用牛刀。
  - 您可以兩者都試用看看，會發現做出來的 Y_pred ，答案一模一樣

隨堂練習：使用二次方訓練與預測

- 請撰寫下列程式碼，並執行看看：

```python
from HappyML.regression import SimpleRegressor
import HappyML.model_drawer as md

model = SimpleRegressor()
model.fit(X_poly, Y)
Y_pred = model.predict(X_poly)

md.sample_model(sample_data=(X, Y), model_data=(X, Y_pred))
```

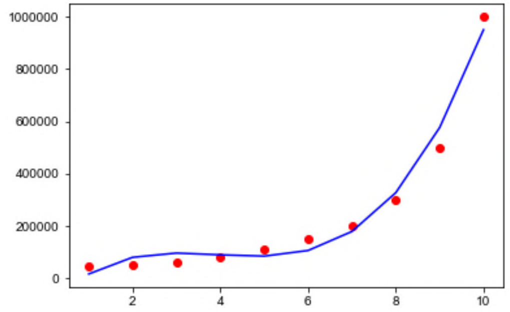
提高次方項、增加精準度

- 還能夠更好嗎？

```
from sklearn.preprocessing import PolynomialFeatures
import pandas as pd
from HappyML.regression import SimpleRegressor
import HappyML.model_drawer as md
from HappyML.performance import rmse

deg = ?
poly_feat = PolynomialFeatures(degree=deg)
X_poly = pd.DataFrame(poly_feat.fit_transform(X))

X_train, X_test, Y_train, Y_test = pp.split_train_test(X_poly, Y, train_size=0.8)

model = SimpleRegressor()
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)

md.sample_model(sample_data=(X, Y), model_data=(X, model.predict(X_poly)))
print(f"Degree={deg}  RMSE={rmse(Y_test, Y_pred):.4f}")
```

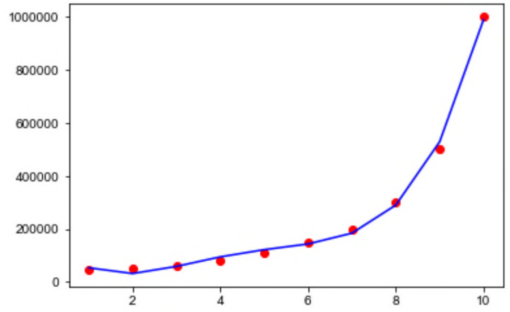
degree = 3

RMSE=38931.5040



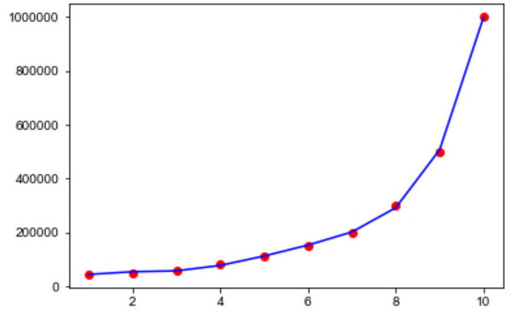
degree = 4

RMSE=14503.2349



degree = 5

RMSE=4047.5035



隨堂練習：提高次方項、增加精準度

- 請修改下列程式碼：
  - 舊程式碼
    - poly_reg = PolynomialFeatures(degree=2)
  - 新程式碼
    - deg=2
    - poly_reg = PolynomialFeatures(degree=deg)
- 請增加下列程式碼，重新切分訓練集、測試集，並計算模型效能：
  - X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X_poly, y_ary=Y, train_size=0.8)
  - print(f"Degree={deg} RMSE={rmse(Y_test, Y_predict):.4f}")
- 將 deg 變數，從 2, 3, 4, 5, ... 一直調整上去，並觀察每次執行後的 RMSE 值。
- 直到上調 deg 值，RMSE 效能指標卻變大變差時，停止整個流程。

使用「快樂版」實作

原始碼解說（1）

引入必要的套件

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
import numpy as np

class PolynomialRegressor:
```

類別的成員變數

```
    __degree = 1
    __regressor = None
    __poly_regressor = None
    __X_poly = None
```

建構函數（預留）

```
    def __init__(self):
        pass
```

degree 成員變數的
getter 與 setter

```
    @property
    def degree(self):
        return self.__degree

    @degree.setter
    def degree(self, degree):
        if degree > 0:
            self.__degree = degree
        else:
            self.__degree = 1
```

原始碼解說（2）

其它成員變數
的 getter

```
    @property
    def X_poly(self):
        return self.__X_poly

    @property
    def regressor(self):
        return self.__regressor

    @property
    def poly_regressor(self):
        return self.__poly_regressor
```

使用 SimpleRegressor
的訓練函數 fit()

```
    def fit(self, x_train, y_train):
        self.__poly_regressor = PolynomialFeatures(self.degree)
        self.__X_poly = self.__poly_regressor.fit_transform(x_train)
        self.__regressor = SimpleRegressor()
        self.__regressor.fit(self.X_poly, y_train)

        return self
```

使用 SimpleRegressor
的預測函數 predict()

```
    def predict(self, x_test):
        x_test = self.__poly_regressor.fit_transform(x_test)
        return self.__regressor.predict(x_test=x_test)
```

原始碼解說（3）

傳入需要的陣列　　最多算到幾次方　　是否顯示過程

```python
    def best_degree(self, x_train, y_train, x_test, y_test, max_degree=10, verbose=False):
        the_best = []
        best_deg, min_rmse = 0, float("inf")

        # Calculate the RMSE of each degree
        for deg in range(1, max_degree+1):
            self.degree = deg
            y_pred = self.fit(x_train, y_train).predict(x_test=x_test)
            this_rmse = np.sqrt(mean_squared_error(y_test, y_pred))

            if this_rmse < min_rmse:
                best_deg = deg
                min_rmse = this_rmse

            the_best.append(best_deg)

            if verbose:
                print("Degree {}: RMSE={:.4f} (BEST DEG={}, RMSE={:.4f})".format(deg, this_rmse, best_deg, min_rmse))
```

← 每回的最佳 degree [1, 2, 3, 4, 5, 5, 5, 8, 9, 10]

← 本次最佳 degree & RMSE 值

迭代各 degree RMSE →

計算這次的 RMSE →

若最佳，則保存 →

保存這次最佳 degree →

印出這次＆最佳效能 →

先看一下執行結果：

```
Degree 1: RMSE=188470.6358 (BEST DEG=1, RMSE=188470.6358)
Degree 2: RMSE=150527.6167 (BEST DEG=2, RMSE=150527.6167)
Degree 3: RMSE=116160.1083 (BEST DEG=3, RMSE=116160.1083)
Degree 4: RMSE=95888.4087 (BEST DEG=4, RMSE=95888.4087)
Degree 5: RMSE=44615.2503 (BEST DEG=5, RMSE=44615.2503)
Degree 6: RMSE=64914.4740 (BEST DEG=5, RMSE=44615.2503)
Degree 7: RMSE=88459.0472 (BEST DEG=5, RMSE=44615.2503)
Degree 8: RMSE=18653.7016 (BEST DEG=8, RMSE=18653.7016)
Degree 9: RMSE=3943.8592 (BEST DEG=9, RMSE=3943.8592)
Degree 10: RMSE=2234.7432 (BEST DEG=10, RMSE=2234.7432)
Frequency vs. Degree dictionary: {1: [1, 2, 3, 4, 8, 9, 10], 3: [5]}
The Best Degree: 5  Frequency: 3
```

原始碼解說（4）

計算最佳 degree 出現次數 →

```python
# Get the best degree
keys_degree, values_freq = np.unique(the_best, return_counts=True)
degree_freq_dict = dict(zip(keys_degree, values_freq))
freq_degree_dict = {}
for k, v in degree_freq_dict.items():
    freq_degree_dict[v] = freq_degree_dict.get(v, [])
    freq_degree_dict[v].append(k)
max_freq = max(freq_degree_dict)
best_deg = max(freq_degree_dict[max_freq])

if verbose:
    print("Frequency vs. Degree dictionary:", freq_degree_dict)
    print("The Best Degree: {}  Frequency: {}".format(best_deg, max_freq))

self.degree = best_deg
return self.degree
```

[1, 2, 3, 4, 5, 5, 5, 8, 9, 10]
→ {1:1, 2:1, 3:1, 4:1, 5:3, 8:1, 9:1, 10:1}

將 key, value 反轉 →

{1:1, 2:1, 3:1, 4:1, 5:3, 8:1, 9:1, 10:1}
→ {1:[1, 2, 3, 4, 5, 9, 10], 3:[5]}

取出出現最多次＆最大 degree →

{1:[1, 2, 3, 4, 5, 9, 10], 3:[5]}

max_freq best_degree

印出最佳 degree & 出現頻率 →

保存最佳 degree，下次fit() 時使用 →

呼叫快樂版函式庫

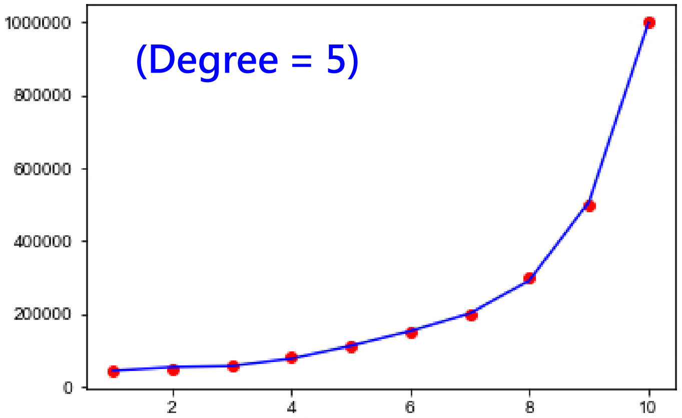
- **呼叫範例：**

計算最佳 degree
並保存在物件肚子裡 →

```python
from HappyML.regression import PolynomialRegressor
import HappyML.model_drawer as md

model = PolynomialRegressor()
model.best_degree(x_train=X_train, y_train=Y_train, x_test=X_test, y_test=Y_test, verbose=True)
Y_pred = model.fit(x_train=X, y_train=Y).predict(x_test=X)

md.sample_model(sample_data=(X, Y), model_data=(X, Y_pred))
print(f"Fit completed! The best degree is {model.degree}")
```

- **執行結果：**

```
Degree 1: RMSE=174932.9678 (BEST DEG=1, RMSE=174932.9678)
Degree 2: RMSE=57037.8744 (BEST DEG=2, RMSE=57037.8744)
Degree 3: RMSE=41790.6490 (BEST DEG=3, RMSE=41790.6490)
Degree 4: RMSE=17067.8502 (BEST DEG=4, RMSE=17067.8502)
Degree 5: RMSE=3659.2108 (BEST DEG=5, RMSE=3659.2108)
Degree 6: RMSE=6045.3290 (BEST DEG=5, RMSE=3659.2108)
Degree 7: RMSE=4768.0392 (BEST DEG=5, RMSE=3659.2108)
Degree 8: RMSE=4312.8944 (BEST DEG=5, RMSE=3659.2108)
Degree 9: RMSE=3694.6311 (BEST DEG=5, RMSE=3659.2108)
Degree 10: RMSE=2349.0994 (BEST DEG=10, RMSE=2349.0994)
Frequency vs. Degree dictionary: {1: [1, 2, 3, 4, 10], 5: [5]}
The Best Degree: 5  Frequency: 5
Fit completed! The best degree is 5
```



隨堂練習：使用快樂版實作

- 請先看懂講師提供的 PolynomialRegressor 類別內的程式碼

- 請輸入下列程式碼，並執行看看：

```python
from HappyML.regression import PolynomialRegressor
import HappyML.model_drawer as md

model = PolynomialRegressor()
model.best_degree(x_train=X_train, y_train=Y_train, x_test=X_test, y_test=Y_test, verbose=True)
Y_pred = model.fit(x_train=X, y_train=Y).predict(x_test=X)

md.sample_model(sample_data=(X, Y), model_data=(X, Y_pred))
print(f"Fit completed! The best degree is {model.degree}")
```

課後作業：預測發電機失效時間

- 說明：
  - 有一款老舊發電機，根據過往統計，「**使用年份**」與「**總失效時間**」如 Device_Failure.csv 所示。

	A	B
1	years_in_use	failure_hours
2	1	3
3	2	5
4	3	7
5	4	18
6	5	43
7	6	85
8	7	91
9	8	98
10	9	100
11	10	130
12	11	230
13	12	487

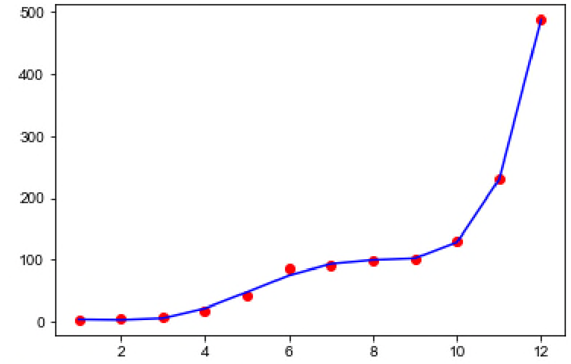
  - 請撰寫一個程式，能讓使用者輸入「**目前使用年份**」，並預測「**總失效時間**」與「**每年平均失效時間**」。

課後作業：預測發電機失效時間

- 提示：
  - 請點擊下列網址 https://bit.ly/3I7nD4O ，下載資料集。
  - 請對該資料集，進行前處理、並訓練出一個多項式迴歸模型。
  - 詢問使用者「**目前使用年份**」。
  - 用多項式迴歸模型，預測出「**預計總失效時間**」、並計算出「**平均每年失效時間**」。
  - 輸出如下圖所示：

```
請輸入設備已使用年份：13
您的設備預測總失效時間 = 1022.4545 小時
平均每年失效時間 = 78.6503 小時/年
```

  - 最後要能印出「樣本點」與「模型」的**適配程度**、使用**幾次方**的「多項式迴歸」、以及模型的 **RMSE** 值。如下所示：



Degree = 5 RMSE = 3.6687

重點整理：多項式迴歸

- 定義
  - 可用 $c_0 + c_1 X^1 + \cdots c_n X^n$ 擬合出模型的問題
  - 通常為「**一元 N 次方程式**」

- 相關外掛套件
  - 增加次方倍： sklearn.preprocessing.PolynomialFeatures
  - 擬合與測試： 可用 LinearRegression() 或 statsmodels 下的 OLS()