Fourth Industrial Summer School

Module 4: ML

Supervised Learning: Regression

Friday, July 5, 2024

Outlines

✓ Complex/flexible models

- ✓ Polynomial Regression
- ✓ Stepwise Regression

Flexible Regression

- The relation between the independent and dependent variables is not always linear! Let us suppose the data is quadratic.
- A quadratic regression is the process of finding a model that estimates the parabola and best fits the data.
- The quadratic regression equation is in the form of:

$$y = \beta_0 + \beta_1 x + \boldsymbol{\beta_2 x^2}, \qquad where\ \beta_2 \neq 0$$

	X	Y
0	-3	7.5
1	-2	3.0
2	-1	0.5
3	0	1.0
4	1	3.0
5	2	6.0
6	3	14.0

Quadratic Regression

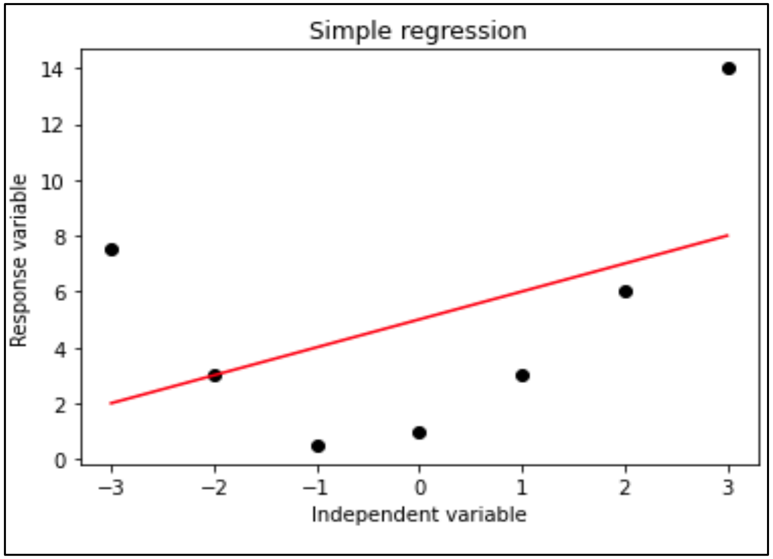
Let us consider the dataset, how can we develop a model for it?

Quadratic Regression

```python
# data
X = np.array([ -3, -2, -1, 0, 1, 2, 3]).reshape(-1,1)
y = np.array([7.5, 3, 0.5, 1, 3, 6, 14])

# modeling and prediction
Lr = LinearRegression().fit(X,y)
y_pred = Lr.predict(X)

# visualization
plt.scatter(X,y, c='k')
plt.plot(X, y_pred, c='r')
plt.title('Simple regression')
plt.xlabel('Independent variable')
plt.ylabel('Response variable')
plt.show()
```



- As we can see, the trained model is simple
- This model implies underfitting the data, so both taring and testing errors will be high.
- We want to develop a more flexible model (in this case, quadratic), let us see how?

Quadratic transformation of the data

- To enforce Sklearn to compute the $\boldsymbol{\beta_2}$, we need to add another column of data (another new variable).
- Sklearn implements of linear regression capable to compute **multi regressions.**
- So, a solution is to extend our single independent variable $\boldsymbol{X^2}$ and create a dataset of two features, also called **vandermonde matrix**.

Original data

Y	X
7.5	-3
3	-2
0.5	-1
1	0
3	1
6	2
14	3

→ Transform →

Transformed data

Y	X	X^2
7.5	-3	9
3	-2	4
0.5	-1	1
1	0	0
3	1	1
6	2	4
14	3	9

Also called **Vandermonde** matrix

Transform to polynomial features

- The steps is showing in the figure, and the data matrix contains two additional columns.
  - A constant column of ones for estimating the intercept, and
  - the quadratic column (last)

```
# import the method from preprocessing
from sklearn.preprocessing import PolynomialFeatures

# setting up the method instance with degree = 2
poly = PolynomialFeatures(degree = 2)

# use fit_transform to transform the data
X2 = poly.fit_transform(X)
X2
```

```
array([[ 1., -3.,  9.],
       [ 1., -2.,  4.],
       [ 1., -1.,  1.],
       [ 1.,  0.,  0.],
       [ 1.,  1.,  1.],
       [ 1.,  2.,  4.],
       [ 1.,  3.,  9.]])
```

- What we need to do now is simply apply ordinary linear regression modeling

```
# modeling and prediction
Lr = LinearRegression().fit(X2,y)
y_pred = Lr.predict(X2)
```

Sklearn: Pipeline

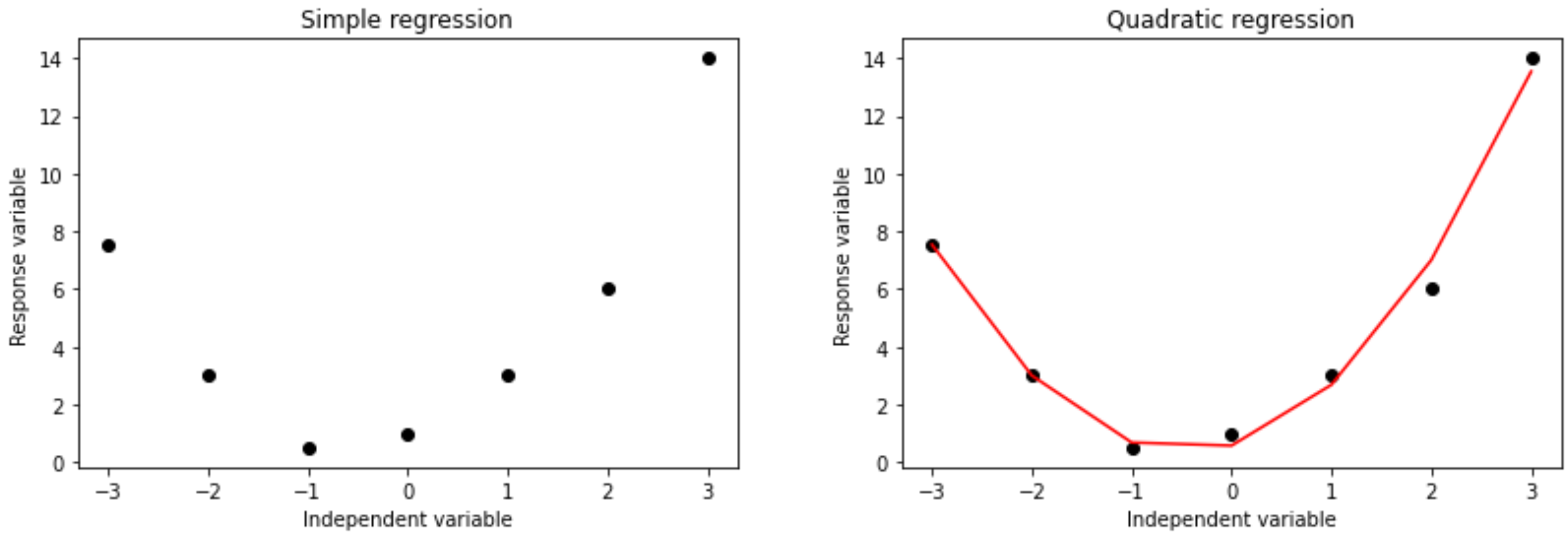
- Let us combine the two steps in the previous slide to develop a pipeline estimator.
- Pipelines ease our objective to build a model by reducing both the coding and human mistakes

```
from sklearn.pipeline import make_pipeline
from sklearn.preporcessing import PolynomialFeatures
# 1. Construct a pipeline
model = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
# 2. fit the composite model to carry preprocessing and build a ML model
model.fit(X,y)
# 3. make predictions
y_pred = model.predict(X)
```

- Sequentially apply a list of transforms and a final estimator. Intermediate steps of pipeline must implement fit and transform methods and the final estimator only needs to implement fit.

Quadratic Regression

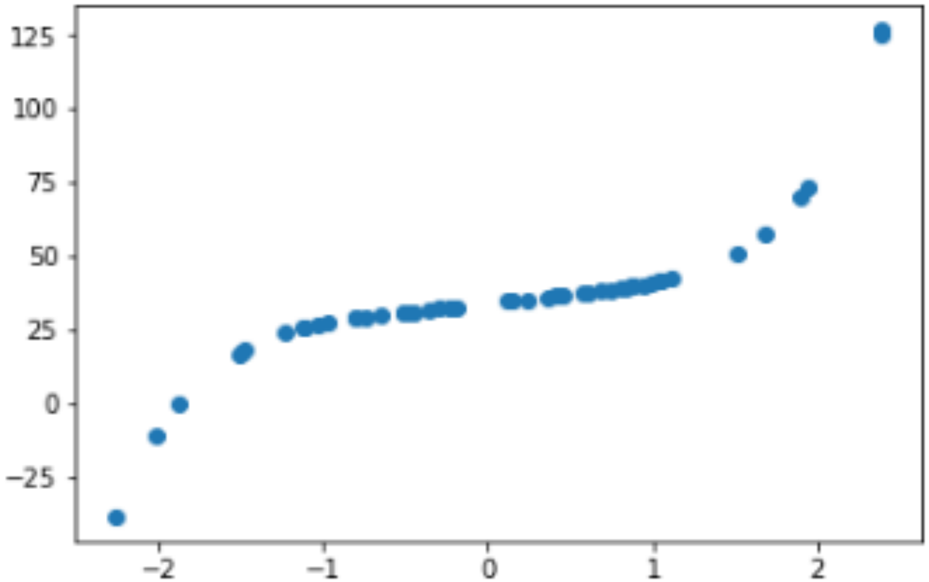
- We see from the results that the new model is better than the simple model that we started with.



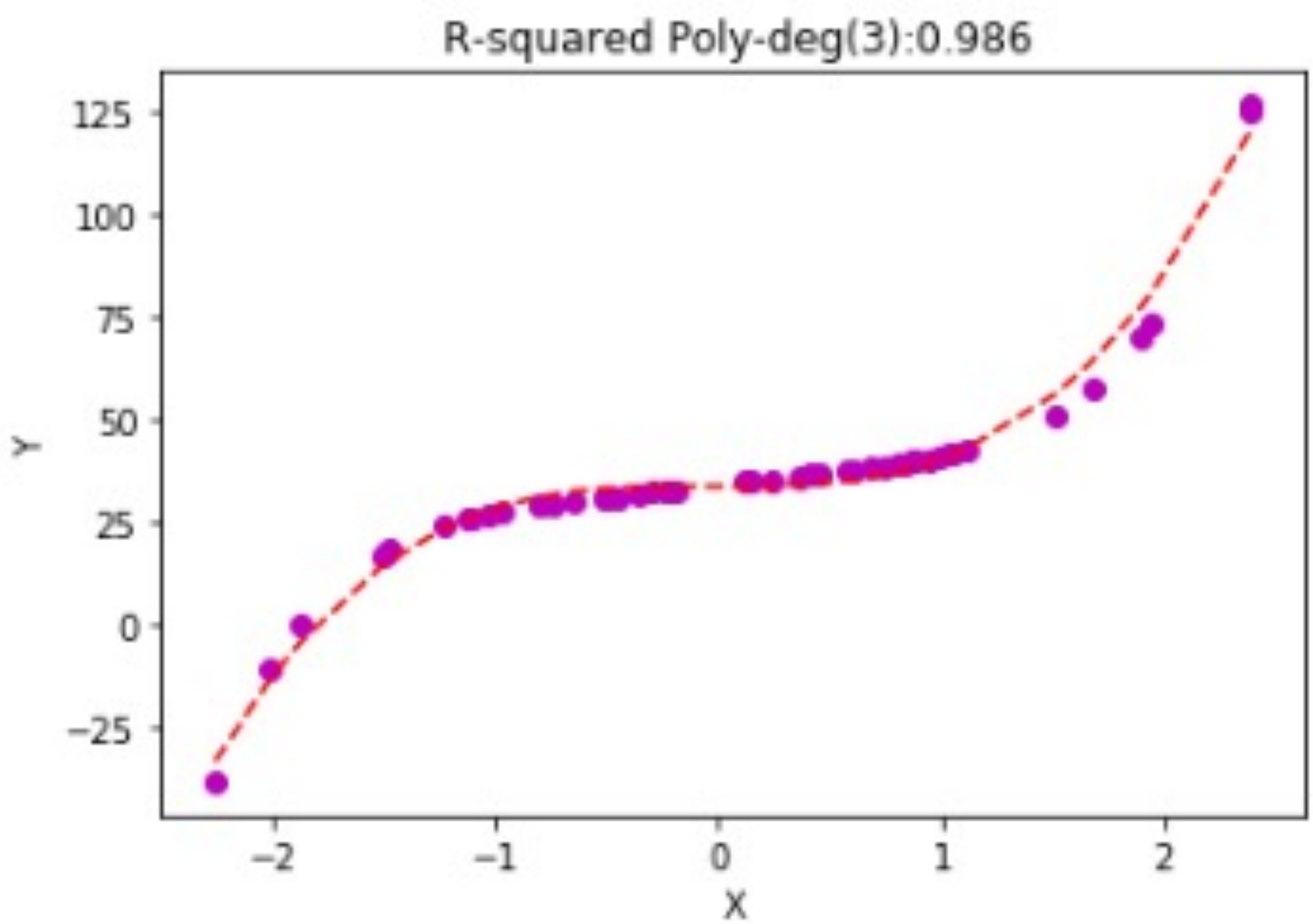
- This procedure can be generalized, and higher-degree features can be computed for **oscillatory datasets**

Polynomial degree regression

- With oscillatory dataset, we can generate several higher degree regression models by looping through a list of values (degrees) 3, 4, 5, 7, .. Then,
- For each model, we compute the R^2 and compare it to the previous model's R^2
- We may stop if the difference is too small and pick the less complex model.



Results from previous example



Piecewise regression

- Unfortunately, polynomial regression has a number of drawbacks:
  - By increasing the complexity of the formula, the number of features is also increased.
  - Polynomial regression models may overfit (high variance).
  - It is inherently non-local, (The fit is affected greatly by any change in Y values during training)

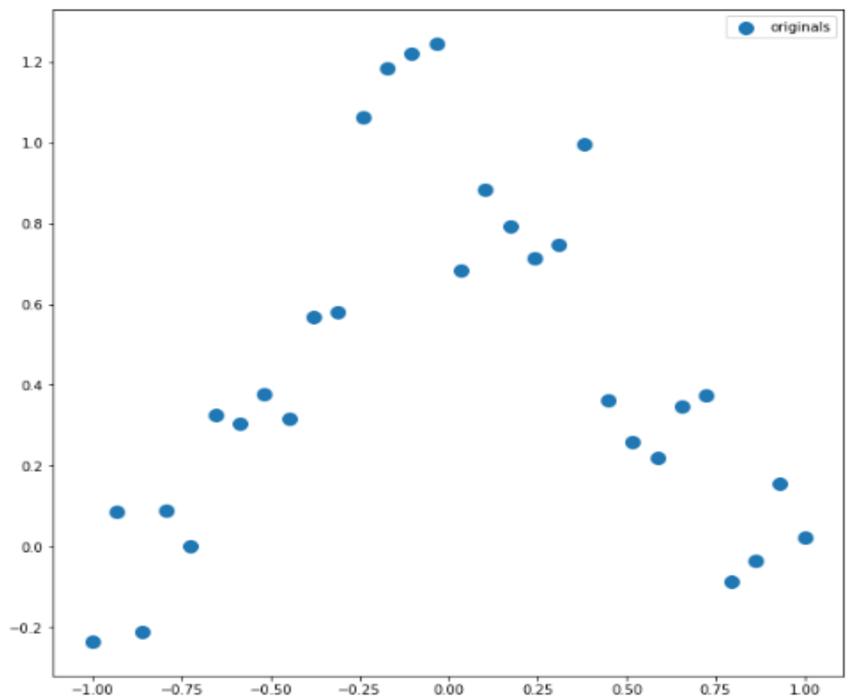
- The complexity imposed by this approach can be substituted with several small degree polynomials, i.e., piecewise regression (splines)

- The disadvantages of the polynomial regression and incompetence of the linear model can be avoided by using piecewise (spline) regression.

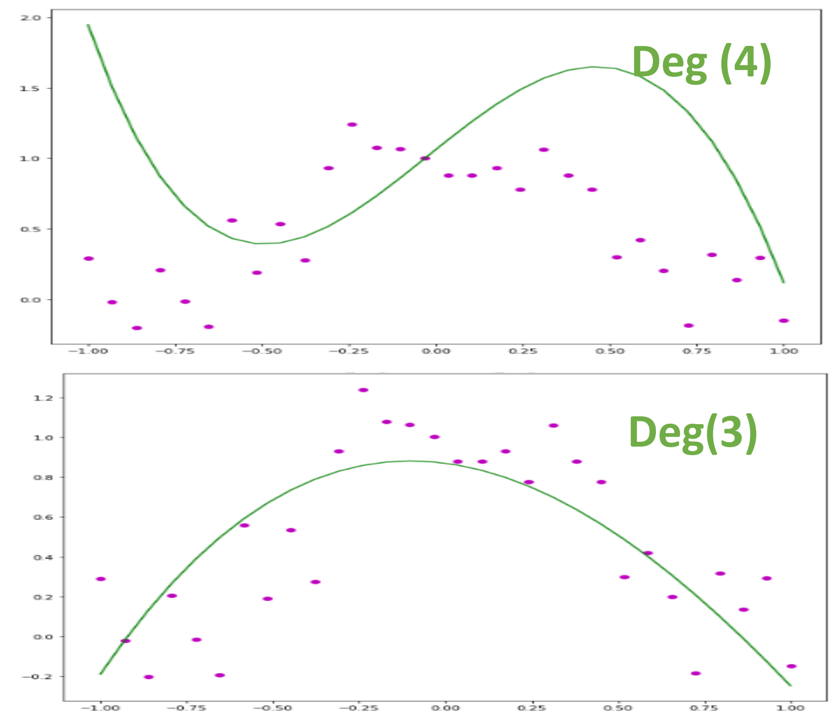
Piecewise polynomial regression

- **An example:** The right figure shows some data generated using polynomial transformation of degree 4, the left figure shows polynomial model with degree (4) plotted over the data

Right



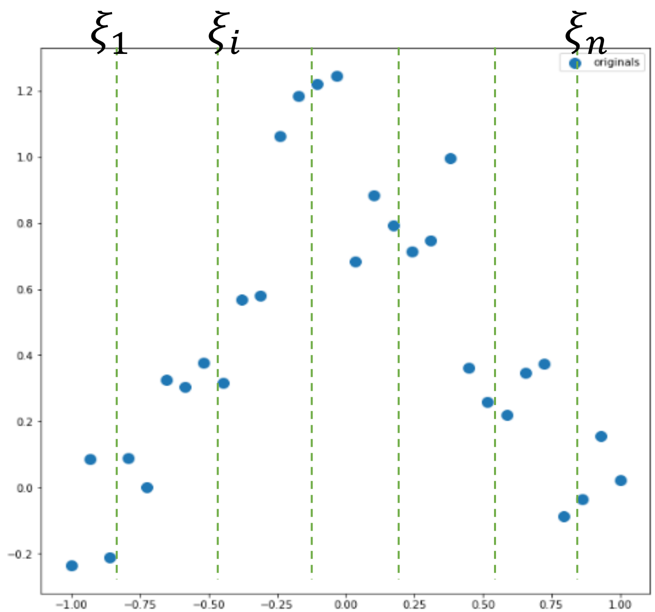
Left

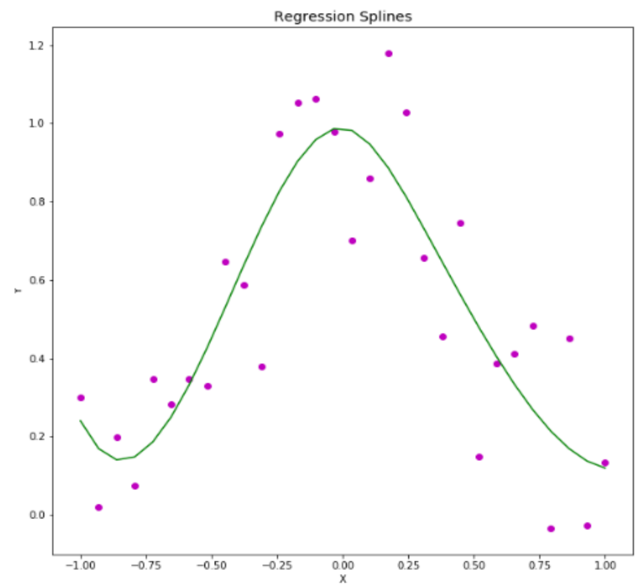


How spline regression works?

- It divides the space into several knots that breaks X into regions.
- Then, with polynomial regression, we can capture the general total trends

$$y_i = \beta 0 + \beta_1 b_1 (x_i) + \beta_2 b_2 (x_i) + \beta_3 b_3 (x_i) + \ldots + \beta_K b_K (x_i)$$





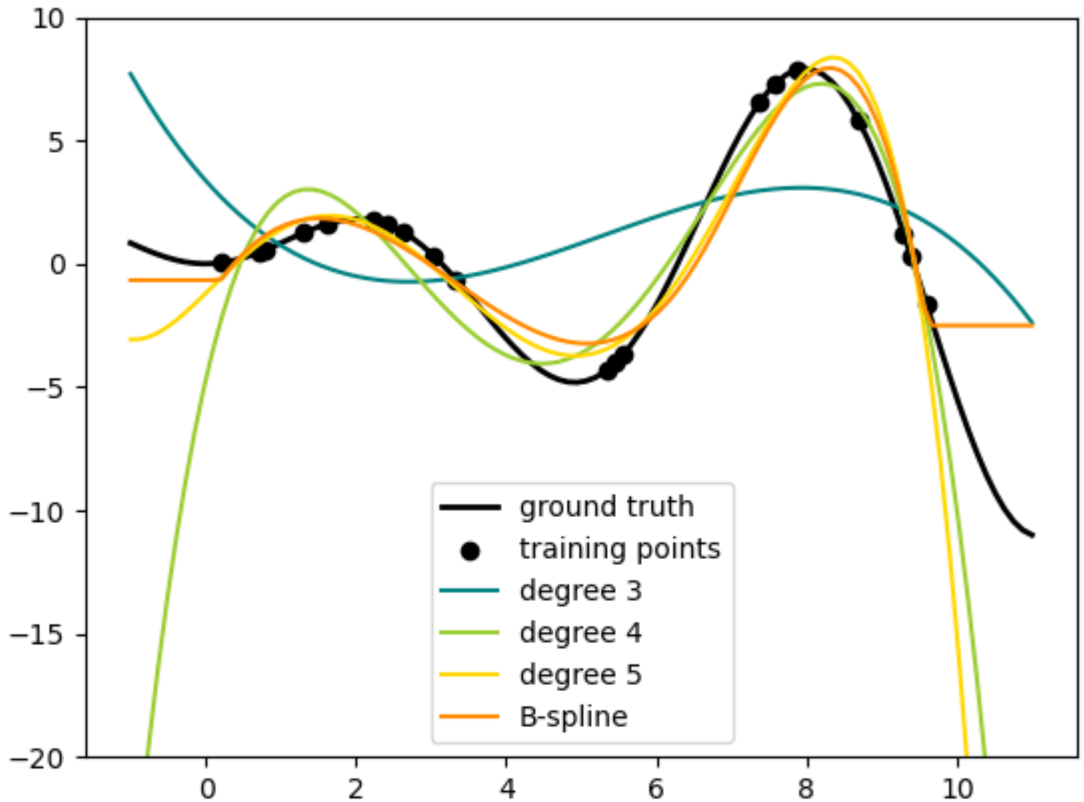
Piecewise regression

- Unlike polynomial regression, which tries to use a polynomial on the whole data to produce flexible fit, splines introduce flexibility locally by defining key regions keeping the degree fixed across all regions.

- Usually, knots are introduced around **rapid changing regions** in the data.

- Rule-of-Thumb: we shouldn't go beyond cubic polynomials with splines (unless one is interested in smoother curves).

Sklearn Example – Comparing Ploy. vs Spline



Source: https://scikit-learn.org/stable/auto_examples/linear_model/plot_polynomial_interpolation.html

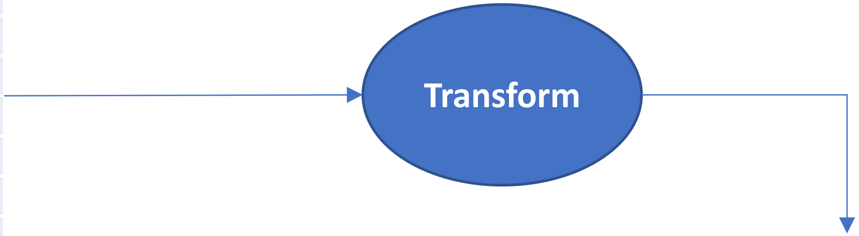
Spline Transform

- Similar to polynomial regression, we start with data transformation to splines
- **Sklearn** provides SplineTransformer to help achieve this task

Original data

Y	X
7.5	-3
3	-2
0.5	-1
1	0
3	1
6	2
14	3

SplineTransformer (n_knots=3, degree=2)




```
array([[0.5       , 0.5       , 0.        , 0.        ],
       [0.22222222, 0.72222222, 0.05555556, 0.        ],
       [0.05555556, 0.72222222, 0.22222222, 0.        ],
       [0.        , 0.5       , 0.5       , 0.        ],
       [0.        , 0.22222222, 0.72222222, 0.05555556],
       [0.        , 0.05555556, 0.72222222, 0.22222222],
       [0.        , 0.        , 0.5       , 0.5       ]])
```

https://mathworld.wolfram.com/CubicSpline.html

Steps to follow

- Similar to polynomial transformation, the same steps can be carried-out

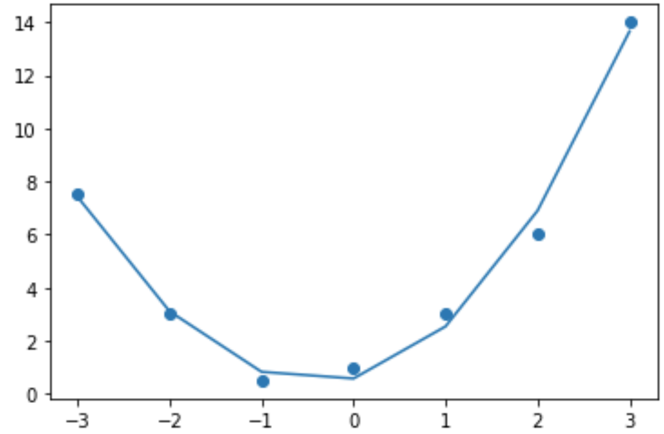
```python
from sklearn.preprocessing import SplineTransformer
from sklearn.linear_model import LinearRegression
# 1. data
y = [7.5, 3, 0.5, 1, 3, 6, 14]; X=np.array([-3, -2, -1, 0, 1, 2, 3])

# 2. Initiate the transformation model and transform
splines = SplineTransformer(degree=2, n_knots=3)
Xs = spline.fit_transform(X.reshape(-1,1))

# 3. build the a regression model
lrs = LinearRegression().fit(Xs, y)

# 4. predict y
y_pred = lrs.predict(Xs)
```

```python
plt.scatter(X, y)
plt.plot(X,y_pred)
plt.show()
```



Exercises (8-9)

- **Nonlinear Regression**
- **Piecewise Regression**