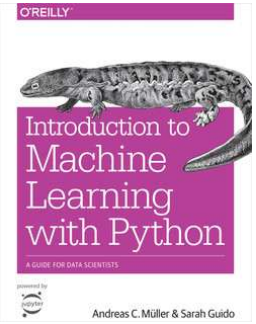
Applied Machine Learning
Polynomial Linear Regression

BSc course Informatiekunde 2026
https://staff.fnwi.uva.nl/a.visser/education/AML

Arnoud Visser
Intelligent Robotics Lab & Computer Vision Lab
Informatics Institute
Universiteit van Amsterdam
A.Visser@uva.nl

Illustrations courtesy of Maarten Marx, Sarah Guido, Yolanda Hagar, and many others.



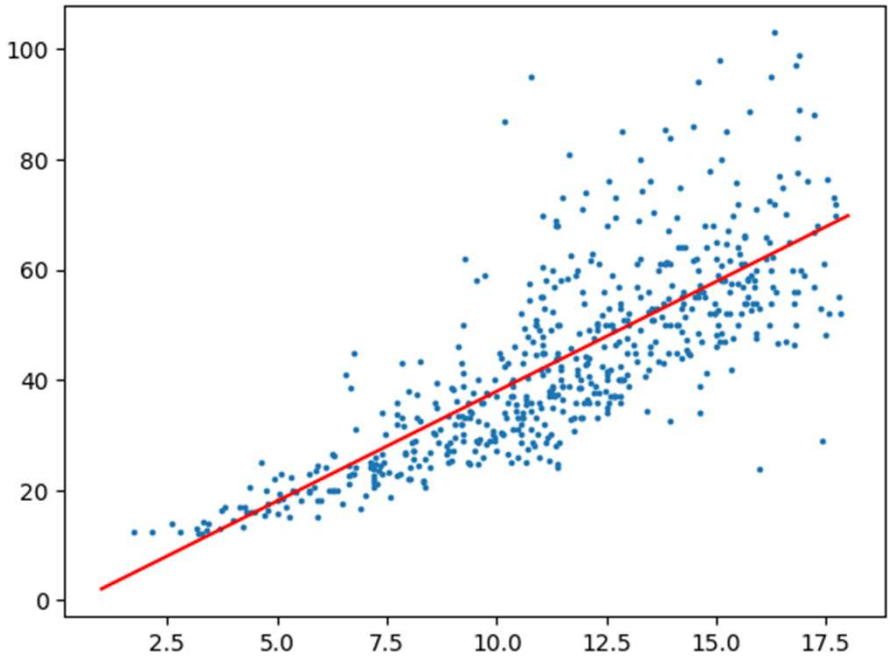
Section 4.4-4.5

Two extremes of modelling

- Modelling 775 datapoints

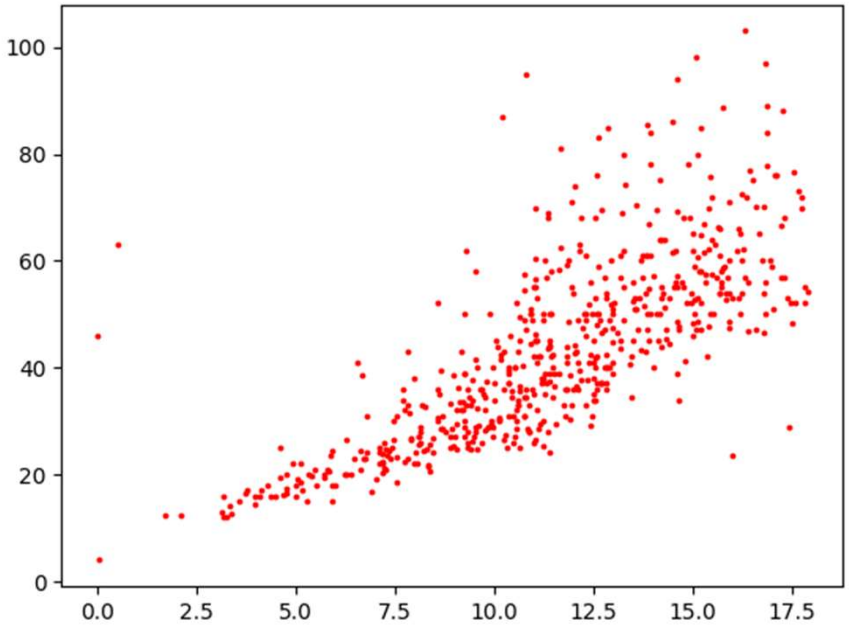
regensburg_pediatric_appendicitis

Linear model



two values ($y = \alpha + \beta x$)

1−nn model



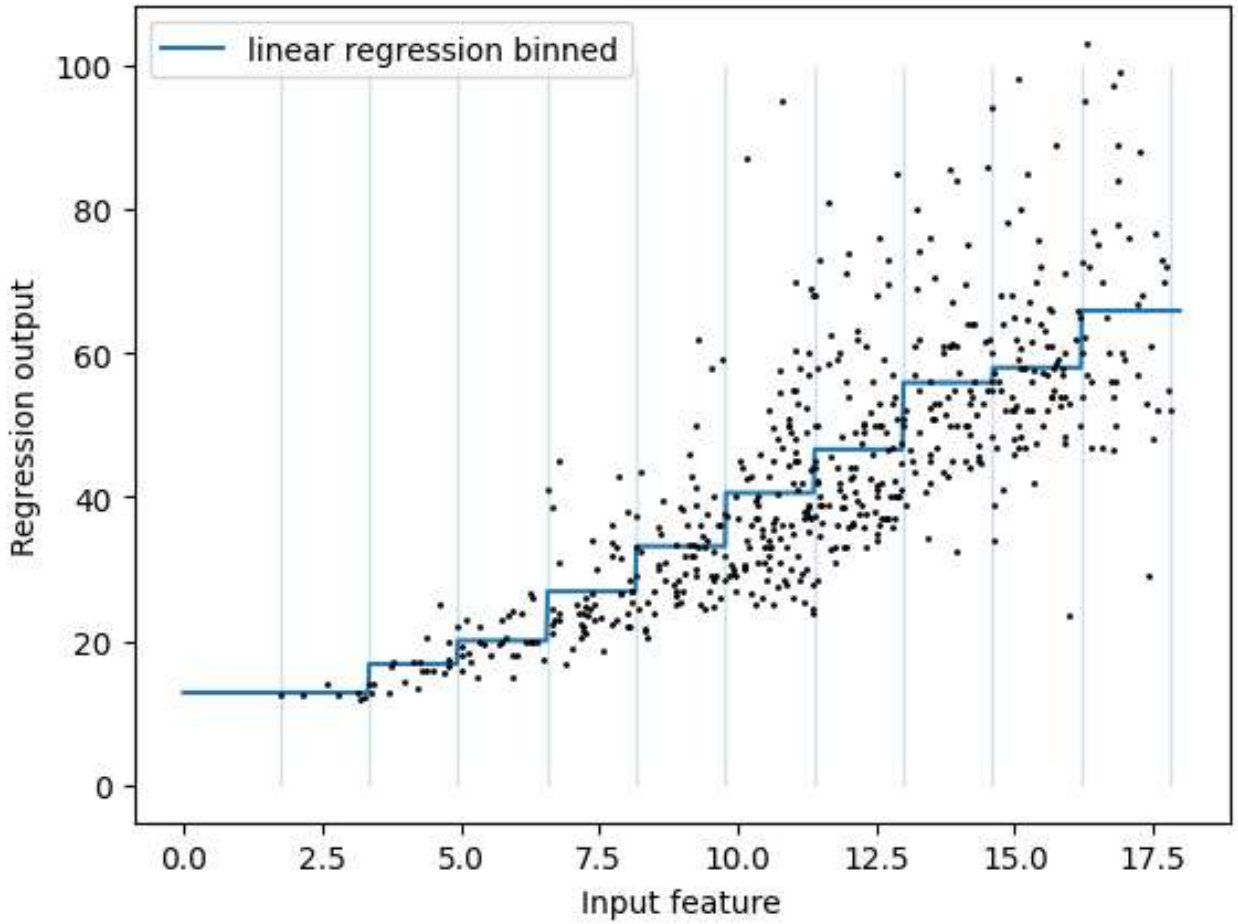
584 training values

BinDiscretizer

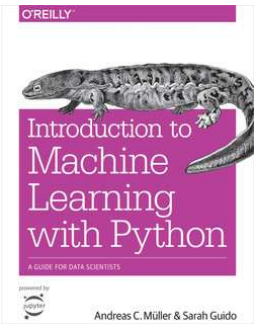
- $weight = averaged\_weight(age \pm 0.9y)$

Model choice

regensburg_pediatric_appendicitis



- 10 discrete bins, uniform distributed



Section 4.4

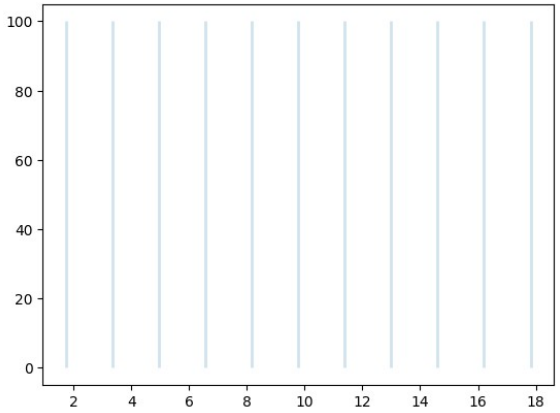
BinDiscretizer

- $weight = averaged\_weight(age \pm 0.9y)$

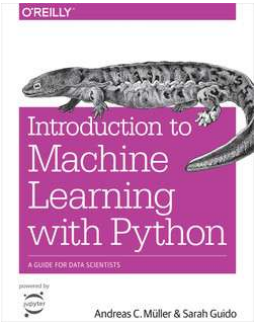
Model implementation

regensburg_pediatric_appendicitis

```
from sklearn.preprocessing import KBinsDiscretizer

kb = KBinsDiscretizer(n_bins=10, strategy='uniform’)

kb.fit(X_train)

plt.vlines(kb.bin_edges_[0], 0, 100, linewidth=1, alpha=.2)
```



- 10 discrete bins, uniform distributed



Section 4.4

(581, 1)

BinDiscretizer

- $weight = averaged\_weight(age \pm 0.9y)$

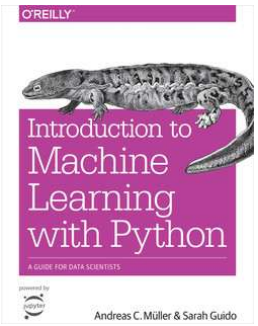
Model implementation

regensburg_pediatric_appendicitis

```
from sklearn.preprocessing import KBinsDiscretizer

kb = KBinsDiscretizer(n_bins=10, strategy='uniform')

kb.fit(X_train)                    (581, 1)

X_binned = kb.transform(X_train)
X_binned
```

```
<581x10 sparse matrix of type '<class 'numpy.float64'>'
        with 581 stored elements in Compressed Sparse Row format>
```

- 10 discrete bins, uniform distributed



Section 4.4

One-Hot encoder

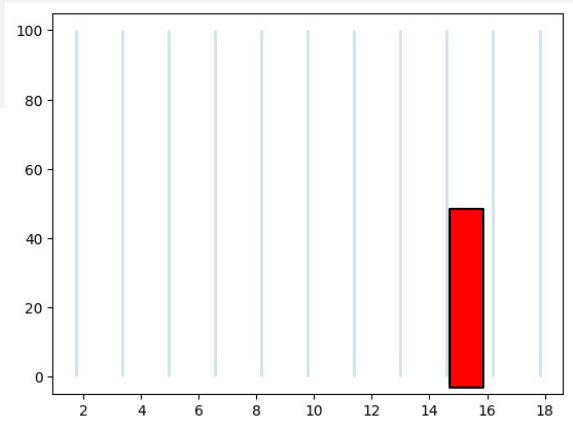
- $weight = averaged\_weight(age \pm 0.9y)$

Model implementation

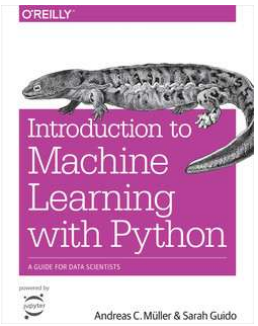
regensburg_pediatric_appendicitis

```
from sklearn.preprocessing import KBinsDiscretizer

kb = KBinsDiscretizer(n_bins=10, strategy='uniform')

kb.fit(X_train)
X_binned = kb.transform(X_train)

X_train[0,0]
X_binned[0,:].toarray()
```

15.89
array([[0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]])



- 10 discrete bins, uniform distributed



Section 4.4

One-Hot encoder

- $weight = averaged\_weight(age \pm 0.9y)$

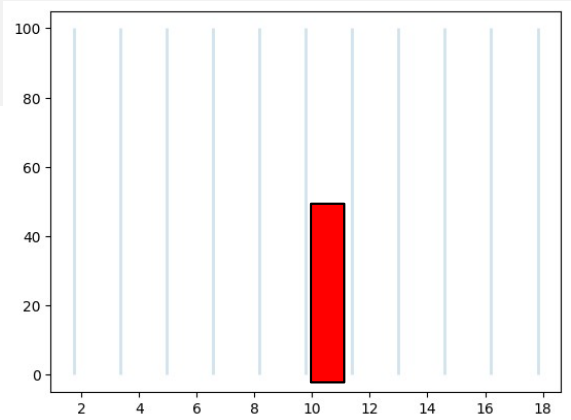
Model implementation

regensburg_pediatric_appendicitis

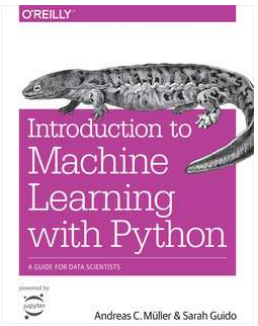
```
from sklearn.preprocessing import KBinsDiscretizer

kb = KBinsDiscretizer(n_bins=10, strategy='uniform')

kb.fit(X_train)
X_binned = kb.transform(X_train)

X_train[1,0]
X_binned[1,:].toarray()
```

11,27
array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]])

(581x)



- 10 discrete bins, uniform distributed



Section 4.4

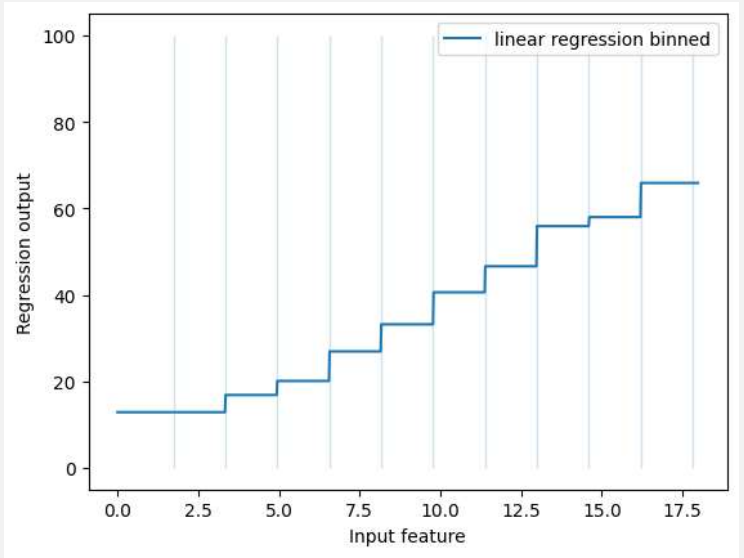
Linear Regression

- $weight = averaged\_weight(age \pm 0.9y)$

Model implementation

regensburg_pediatric_appendicitis

```
from sklearn.preprocessing import KBinsDiscretizer

kb = KBinsDiscretizer(n_bins=10, strategy='uniform')

kb.fit(X_train)
X_binned = kb.transform(X_train)

reg = LinearRegression().fit(X_binned, y_train)

line = np.linspace(0, 18, 1000, endpoint=False).reshape(-1, 1)
line_binned = kb.transform(line)
plt.plot(line, reg.predict(line_binned), label='linear regression binned')
```



Section 4.4

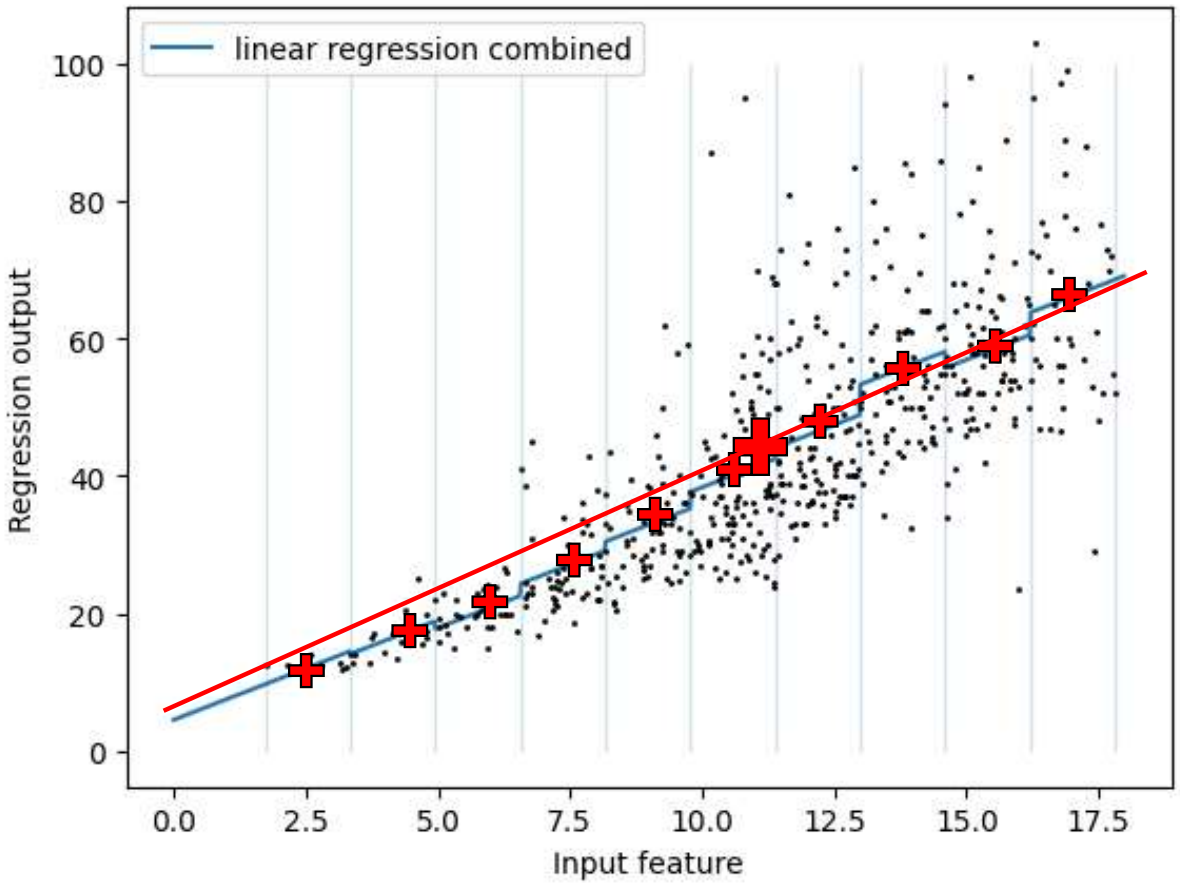
- 10 discrete bins, uniform distributed

Bin & Linear Regression Combined

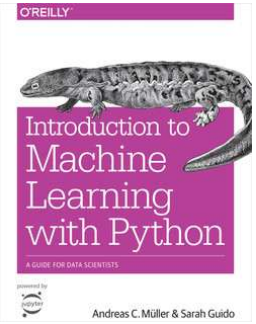
- $weight = averaged_{wei}\quad_{gain} * age + averaged_{weight}(age \pm 0.9y)$

Model choice

regensburg_pediatric_appendicitis



- Original feature + 10 discrete bins



Section 4.5

```
(581, 1) (581, 10) (581, 11)
```

Bin & Linear Regression Combined

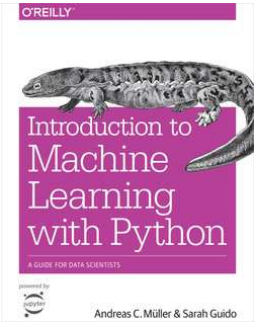
- $weight = averaged_{weight_{gain}} * age + averaged_{weight}(age \pm 0.9y)$

Model implementation

regensburg_pediatric_appendicitis

```
print(X_train.shape)
print(X_binned.shape)
X_combined = np.hstack([X_train, X_binned.toarray()])
print(X_combined.shape)
```

```
(581, 1)
(581, 10)
(581, 11)
```



Section 4.5

- Original feature + 10 discrete bins

(581, 1) (581, 10) (581, 11)

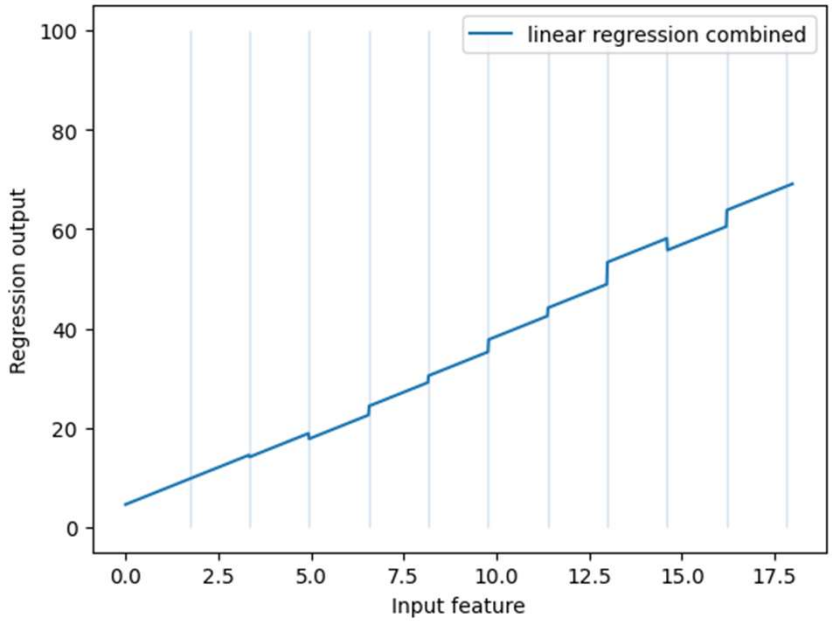
Bin & Linear Regression Combined

- $weight = averaged_{weigh\ gain} * age + averaged_{weight}(age \pm 0.9y)$

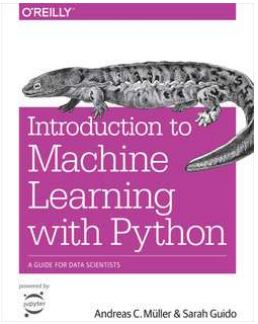
Model implementation

regensburg_pediatric_appendicitis

```
reg = LinearRegression().fit(X_combined, y_train)

line_combined = np.hstack([line, line_binned.toarray()])
plt.plot(line, reg.predict(line_combined), label='linear regression combined')
```



- Original feature + 10 discrete bins



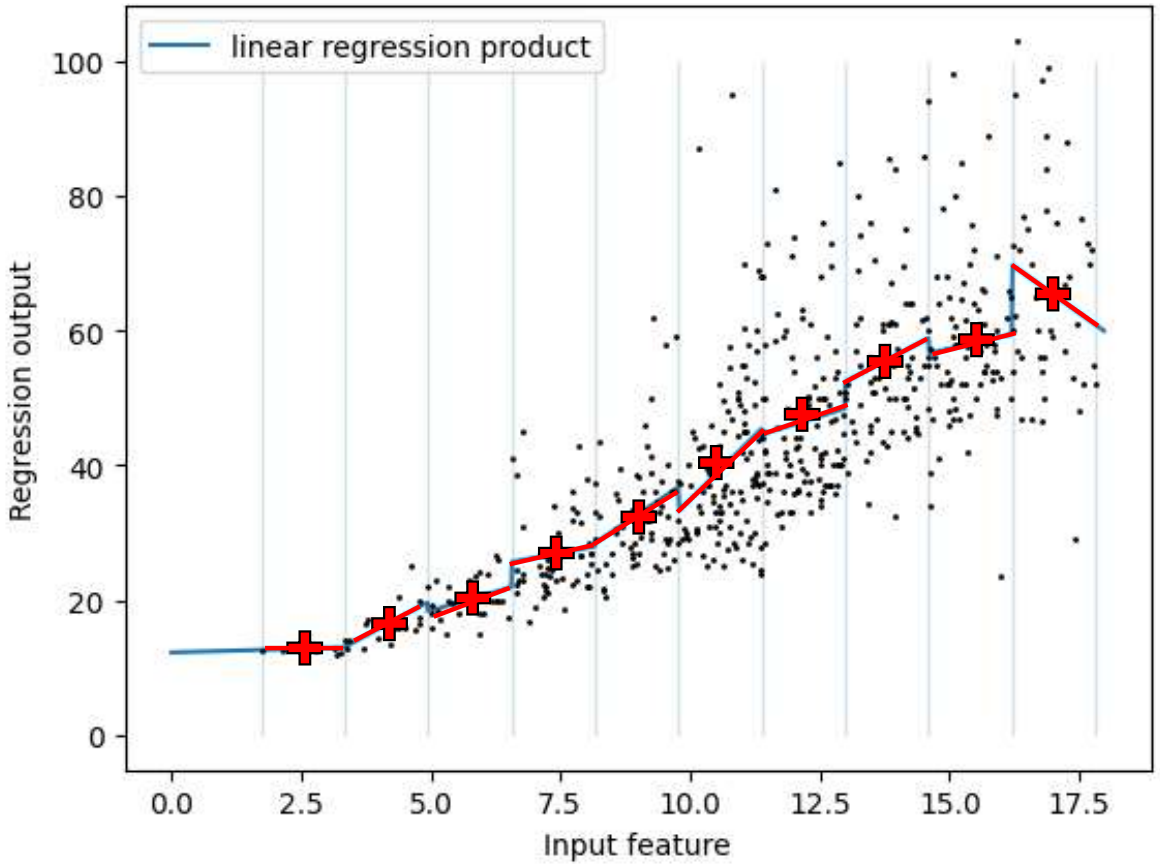
Section 4.5

Linear Regression in each bin

- $weight = averaged_{weig\ \ gain} * age + averaged_{weight}(age \pm 0.9y)$

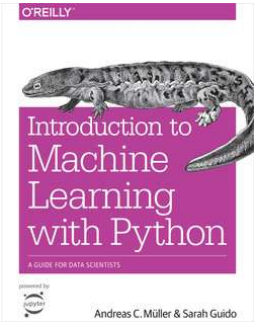
Model choice

regensburg_pediatric_appendicitis



20 values
($y = \alpha + \beta x$)

- 10 discrete bins, each with ($y = \alpha + \beta x$)



Section 4.5

Linear Regression in each bin

- $weight = averaged_{weig\ \ gain} * age + averaged_{weight}(age \pm 0.9y)$

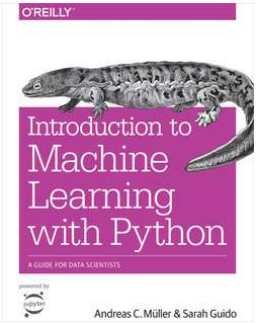
Model implementation

```
print(X_binned.shape)
X_product = np.hstack([X_binned.toarray(), X_train * X_binned.toarray()])
print(X_product.shape)
```

```
(581, 10)
(581, 20)
```

20 values
$(y = \alpha + \beta x)$

- 10 discrete bins, each with $(y = \alpha + \beta x)$



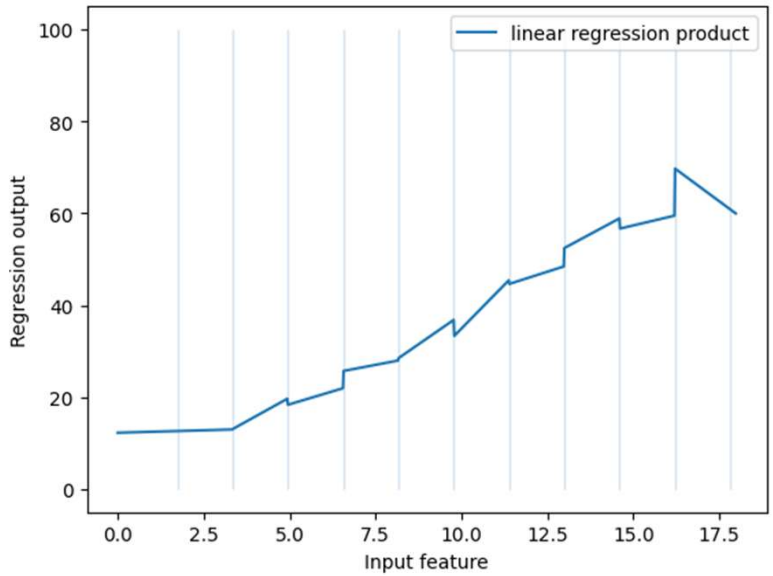
Section 4.5

Linear Regression in each bin

- $weight = averaged_{weight_{gain}} * age + averaged_{weight}(age \pm 0.9y)$

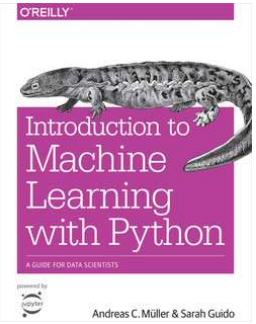
Model implementation

```
reg = LinearRegression().fit(X_product, y_train)

line_product = np.hstack([line_binned.toarray(), line * line_binned.toarray()])
plt.plot(line, reg.predict(line_product), label='linear regression product')
```



20 values
($y = \alpha + \beta x$)

- 10 discrete bins, each with ($y = \alpha + \beta x$)



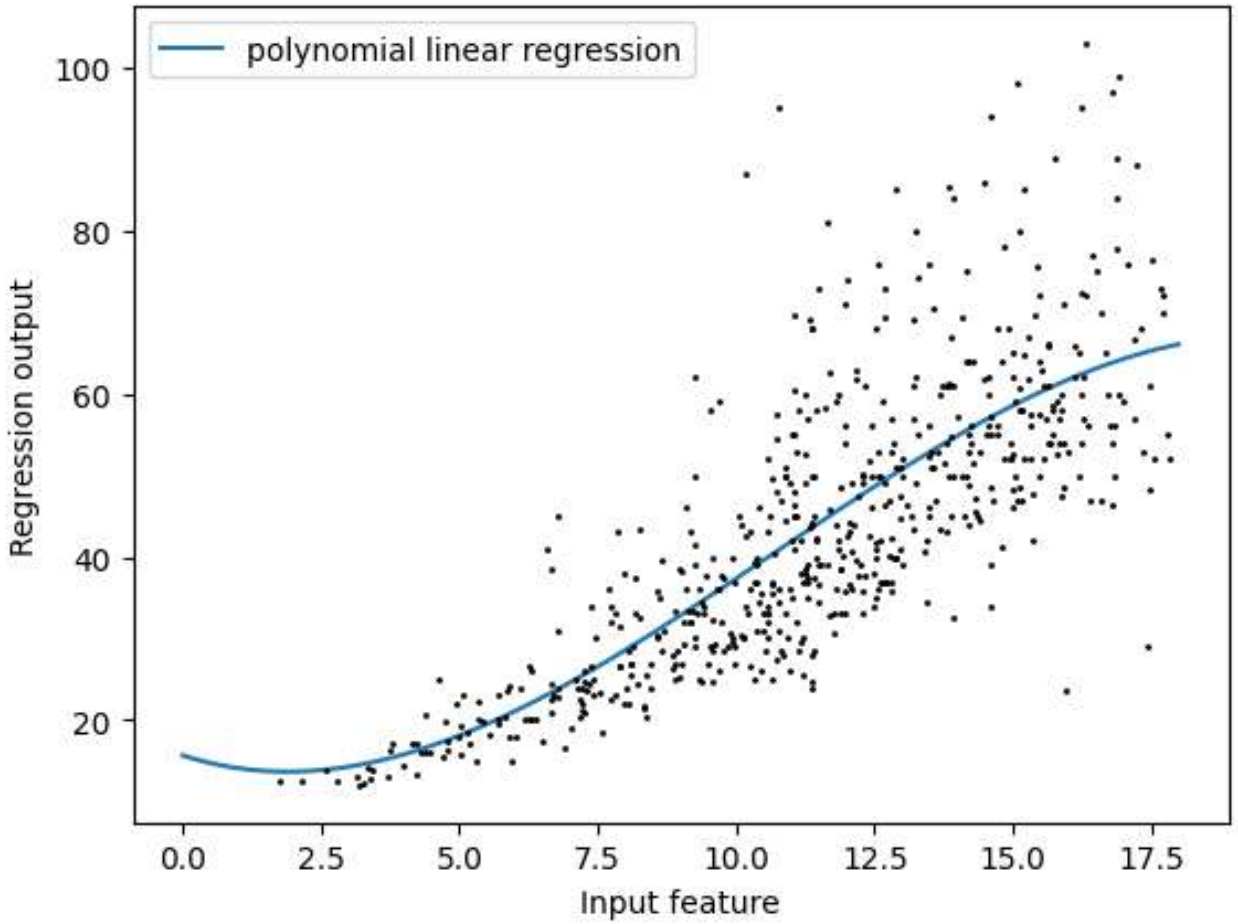
Section 4.5

Polynomial Linear Regression

- $weight = polynomial\_function(age)$

Model choice

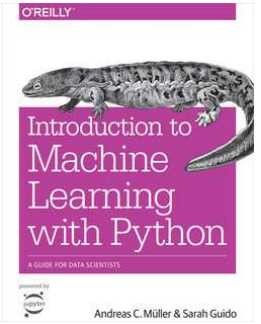
regensburg_pediatric_appendicitis



4 values

$(y = \alpha + \cdots + \delta x^3)$

- 3 functions of x, so $(y = \alpha + \beta x + \gamma x^2 + \delta x^3)$



Section 4.5

Polynomial Linear Regression

- $weight = polynomial\_function(age)$

Model implementation

regensburg_pediatric_appendicitis

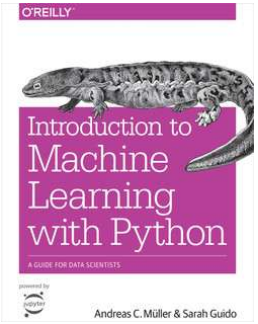
```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=3)
poly.fit(X_train)

X_poly = poly.transform(X_train)

print("X_poly.shape: {}".format(X_poly.shape))
print("Polynomial feature names:\t{}".format(poly.get_feature_names_out()))
```

X_poly.shape: (581, 4)
Polynomial feature names: ['1' 'x0' 'x0^2' 'x0^3']

4 values

$(y = \alpha + \cdots + \delta x^3)$

- 3 functions of x, so $(y = \alpha + \beta x + \gamma x^2 + \delta x^3)$



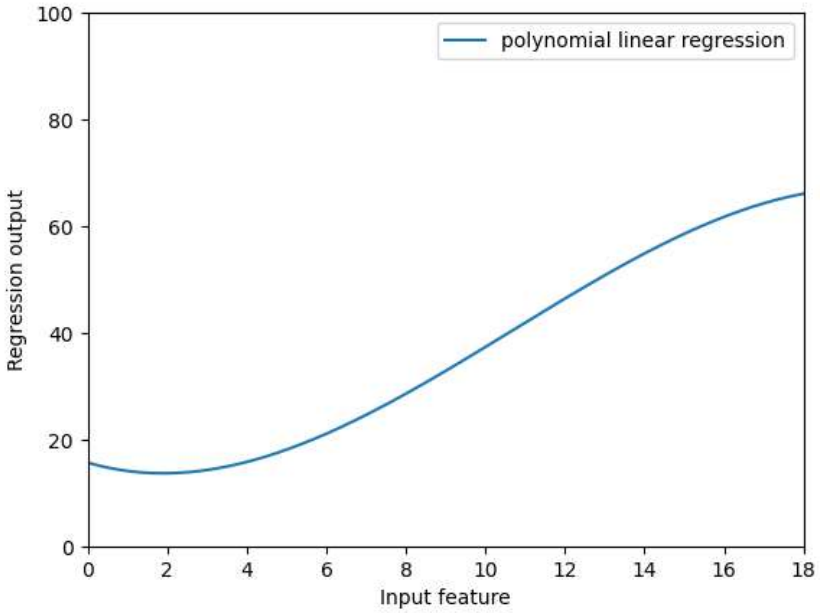
Section 4.5

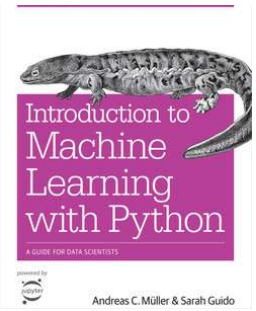
Polynomial Linear Regression

- $weight = polynomial\_function(age)$

Model implementation

```
from sklearn.preprocessing import PolynomialFeature

poly = PolynomialFeatures(degree=3)
poly.fit(X_train)

X_poly = poly.transform(X_train)

reg = LinearRegression().fit(X_poly, y_train)

line_poly = poly.transform(line)
plt.plot(line, reg.predict(line_poly), label='polynomial linear regression')
```



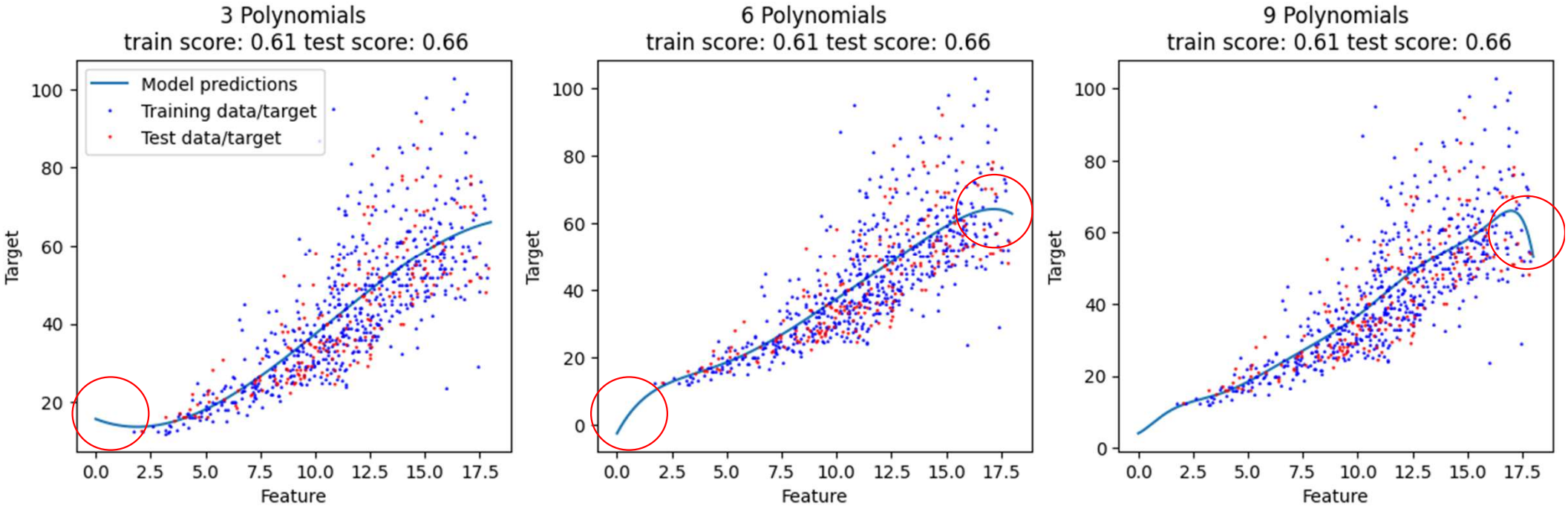


- 3 functions of x, so $(y = \alpha + \beta x + \gamma x^2 + \delta x^3)$

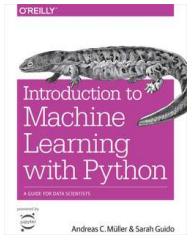
Section 4.5

Higher order polynomials

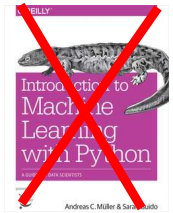
- More freedom on the shape of the function



- Polynomials of a higher degree behave in extreme ways on the boundaries or in regions with little data

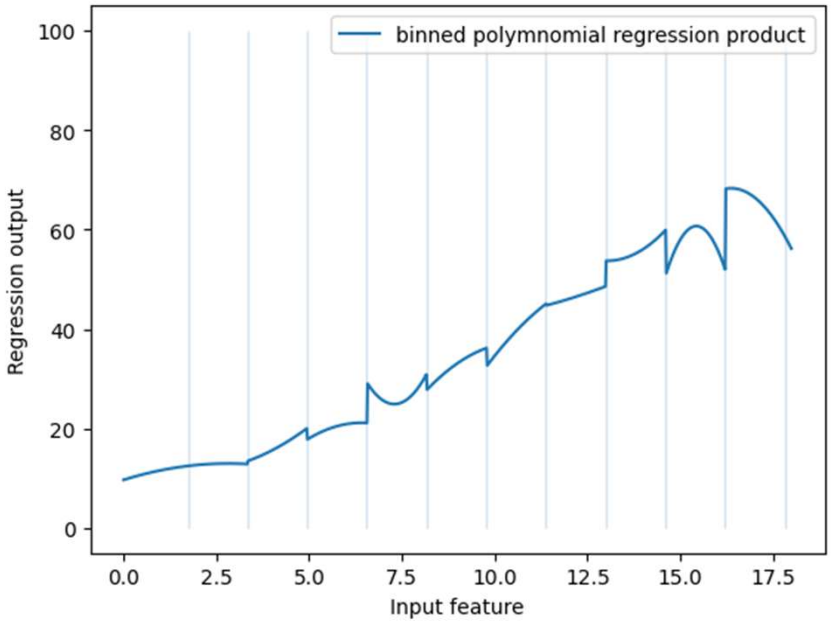


page 234

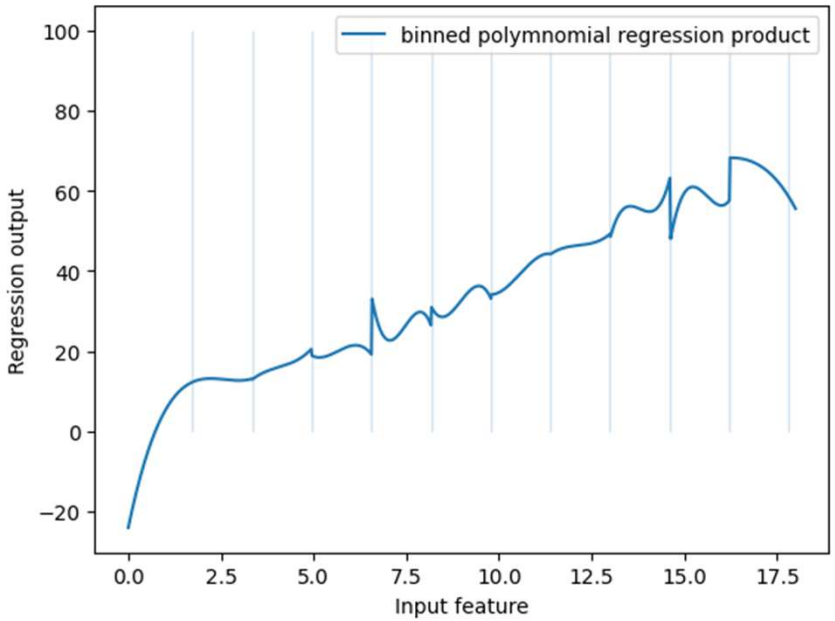


Binned polynomials

- Discretization & Polynomial regression can also be combined



2nd grade



3rd grade

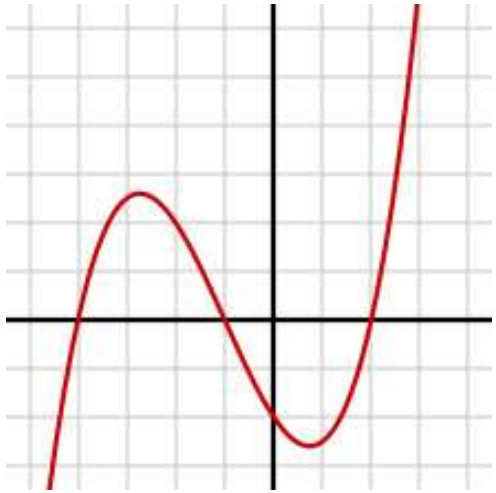
30 values $(y = \alpha + \beta x + \gamma x^2)$

40 values $(y = \alpha + \beta x + \gamma x^2 + \delta x^3)$

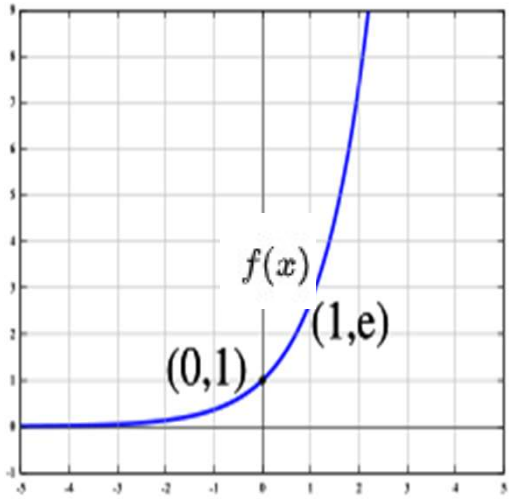
- Not very practical, because of the boundary effects.
  Yet, gives some clues where to expect trouble.

Analytic functions

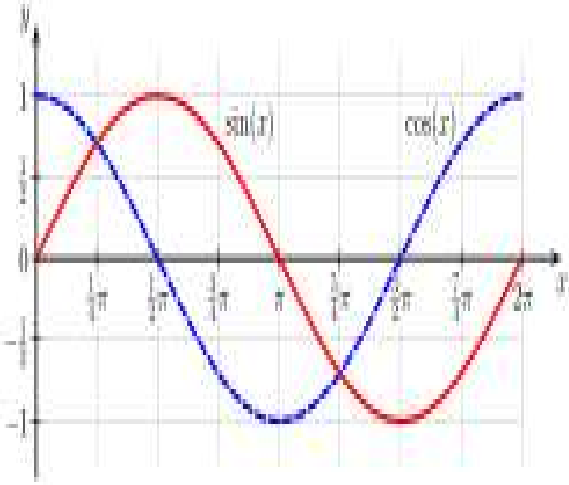
- Smooth functions, functions that are infinitely differentiable, are analytic functions.



Polynomials



Exponential



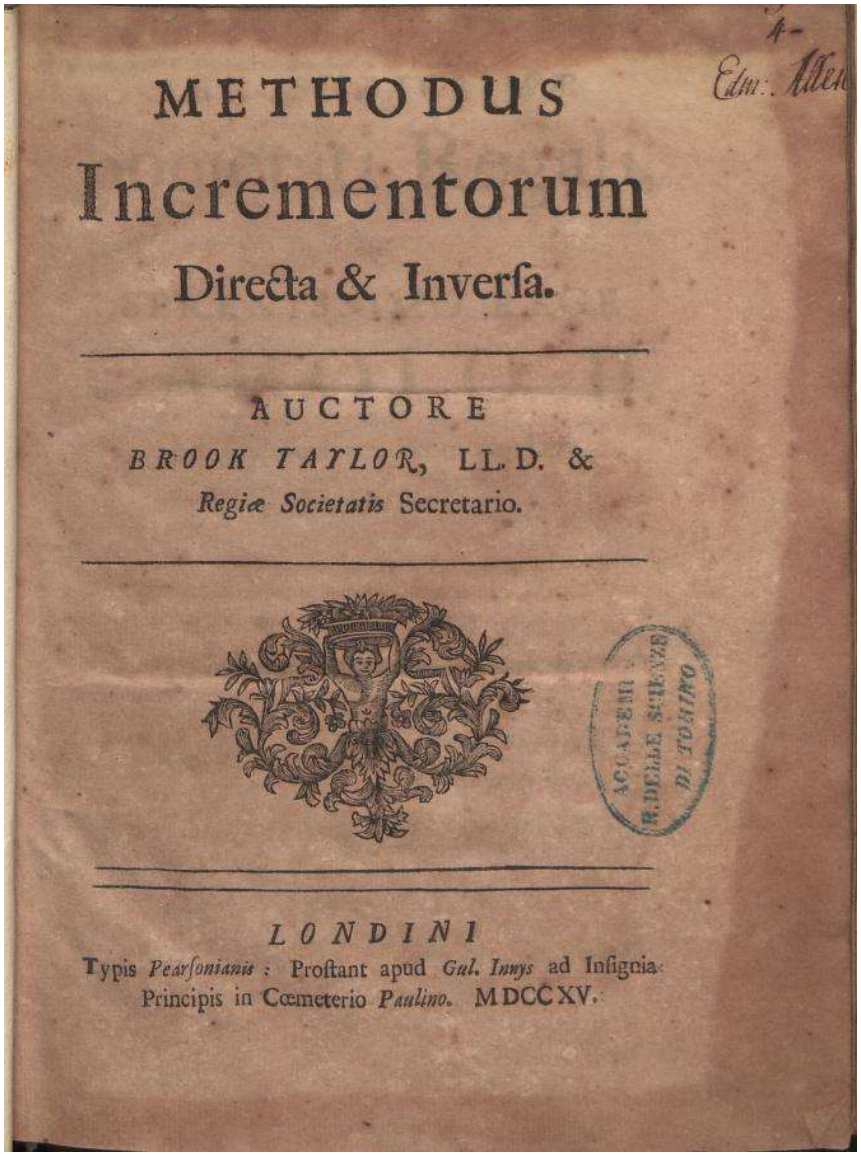
Trigonometric

- An analytic function can be written as infinit sum of basis functions

$$f(x) = \sum_{n=0}^{\infty} a_n (x - x_0)^n$$

Polynomial

Tayler series


METHODUS
Incrementorum
Directa & Inverſa.

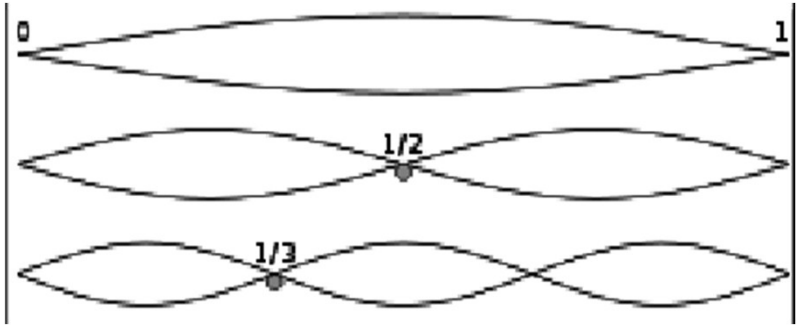
AUCTORE
BROOK TAYLOR, LL.D. &
Regiæ Societatis Secretario.

LONDINI
Typis Pearſonianis: Proſtant apud Gul. Innys ad Inſignia
Principis in Cœmeterio Paulino. MDCCXV.

In the Royal Society of London for Improving Natural Knowledge there were rumours that Isaac Newton had developed a general method for expanding functions in series.

Isaac Newton had shown some examples for the functions $\sin x$ and $\cos x$, but never published it.

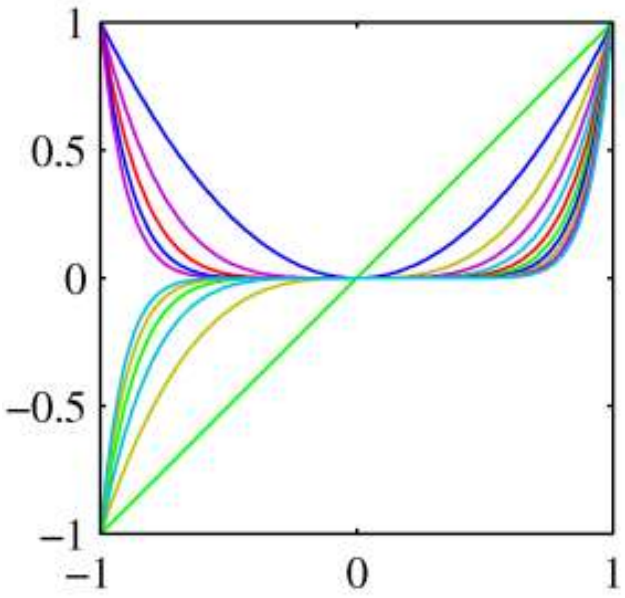
That allowed Brook Taylor to publish the "calculus of finite differences“.



basis functions

$$f(x) = \sum_{n=0}^{\infty} a_n \ \phi_n(x - x_0)$$

- Polynomial basis functions $\phi_n(\boldsymbol{x}) = \boldsymbol{x}^n$



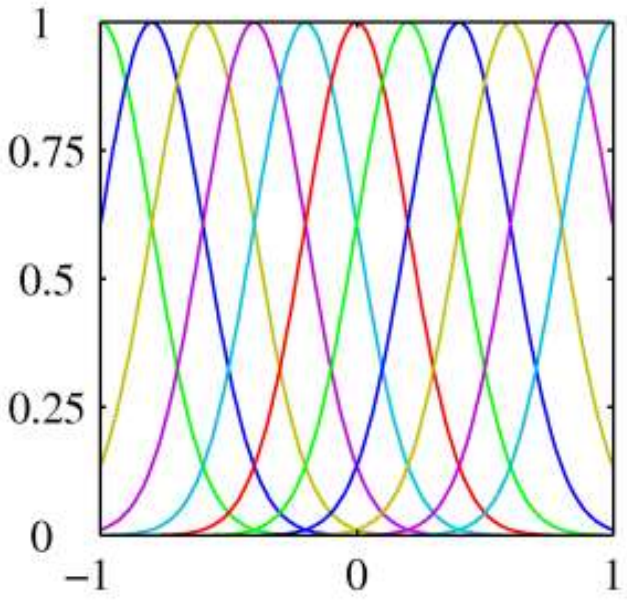
Based on slides from Byron Boots, which was inspired by Christopher Bishop

Other basis functions

$$f(x) = \sum_{n=0}^{\infty} a_n \quad \phi_n(x - x_0)$$

- Gaussian basis functions $\phi_n(\boldsymbol{x}) = e^{-\frac{(x-\mu_n)^2}{2\sigma^2}}$

Mixture of Gaussians



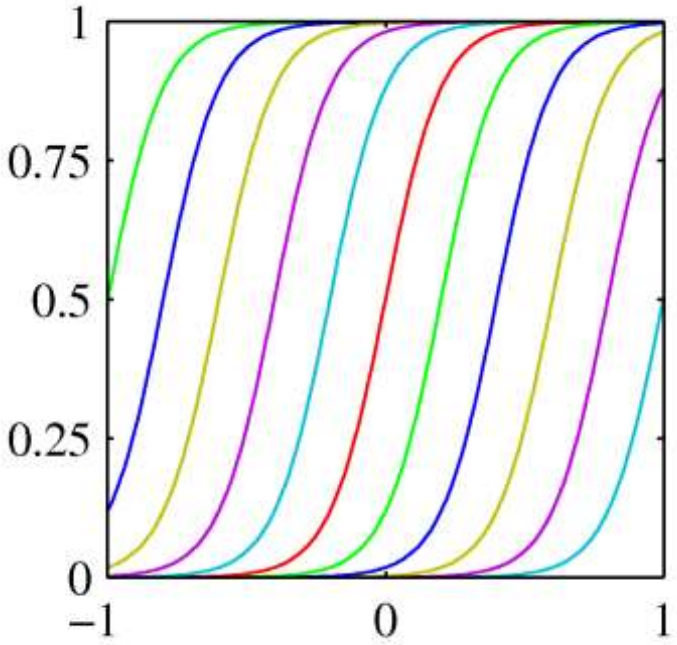
Based on slides from Byron Boots, which was inspired by Christopher Bishop

Other basis functions

$$f(x) = \sum_{n=0}^{\infty} a_n \quad \phi_n(x - x_0)$$

- Sigmoidal basis functions $\phi_n(\boldsymbol{x}) = \varsigma(\frac{\boldsymbol{x}-\mu_n}{\sigma})$ $\zeta(\boldsymbol{x}') = \frac{1}{1+e^{-x\prime}}$

Activation function in NNs



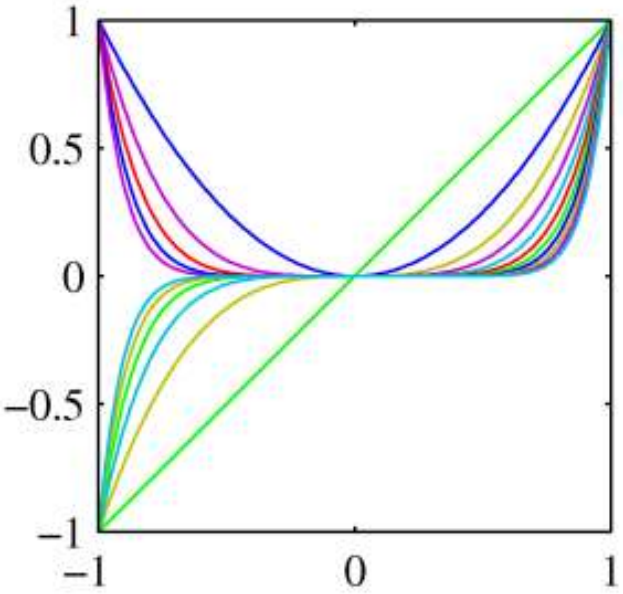
Based on slides from Byron Boots, which was inspired by Christopher Bishop

Other basis functions

$$f(x) = \sum_{n=0}^{\infty} a_n \quad \phi_n(x - x_0)$$

- Polynomial basis functions $\phi_n(\boldsymbol{x}) = \boldsymbol{x}^n$

Global estimate from x_0 :
a small change effects all a_n



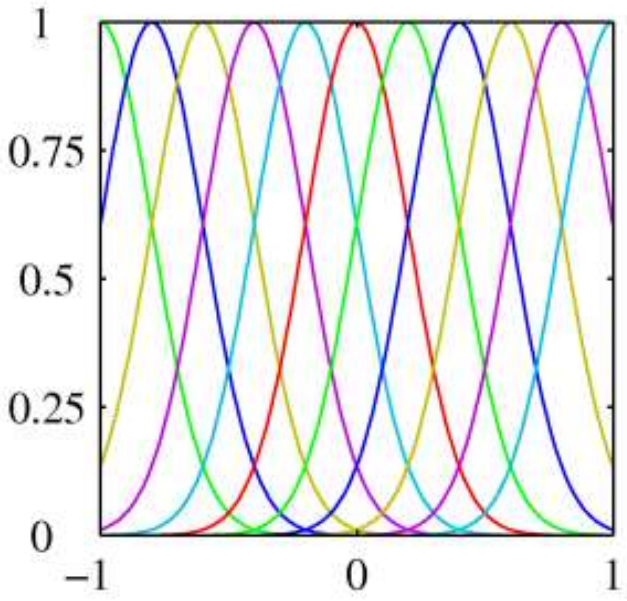
Based on slides from Byron Boots, which was inspired by Christopher Bishop

Other basis functions

$$f(x) = \sum_{n=0}^{\infty} a_n \quad \phi_n(x - x_0)$$

- Gaussian basis functions $\phi_n(\boldsymbol{x}) = e^{-\frac{(x-\mu_n)^2}{2\sigma^2}}$

Local estimate from x_0 :
a small change effects only
nearby basis functions



Based on slides from Byron Boots, which was inspired by Christopher Bishop

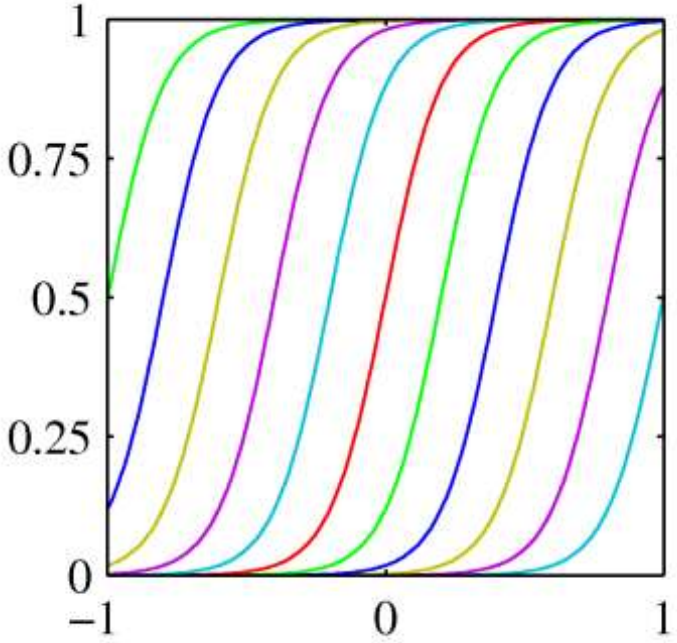
Other basis functions

$$f(x) = \sum_{n=0}^{\infty} a_n \quad \phi_n(x - x_0)$$

- Sigmoidal basis functions $\phi_n(\boldsymbol{x}) = \varsigma(\frac{\boldsymbol{x}-\mu_n}{\sigma})$ $\zeta(\boldsymbol{x}') = \frac{1}{1+e^{-x\prime}}$

Local estimate from x_0 :
a small change effects only
nearby basis functions

Will come back as kernel trick
from
Support Vector Machines



Based on slides from Byron Boots, which was inspired by Christopher Bishop

Fitting a Trigonometric function

- First with Polynomial basis functions

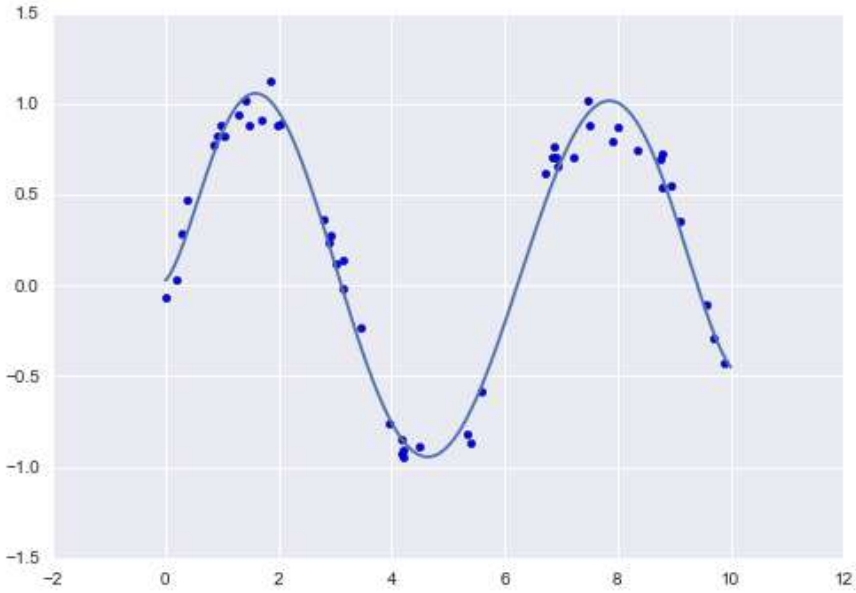
```
rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = np.sin(x) + 0.1 * rng.randn(50)
```

```
from sklearn.pipeline import make_pipeline
poly_model = make_pipeline(PolynomialFeatures(7),
                           LinearRegression())
```

← 7 basic functions are needed!

```
poly_model.fit(x[:, np.newaxis], y)
yfit = poly_model.predict(xfit[:, np.newaxis])
```

```
plt.scatter(x, y)
plt.plot(xfit, yfit);
```



Page 394

Fitting a Trigonometric function

- First with Gaussian basis functions

```
rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = np.sin(x) + 0.1 * rng.randn(50)
```

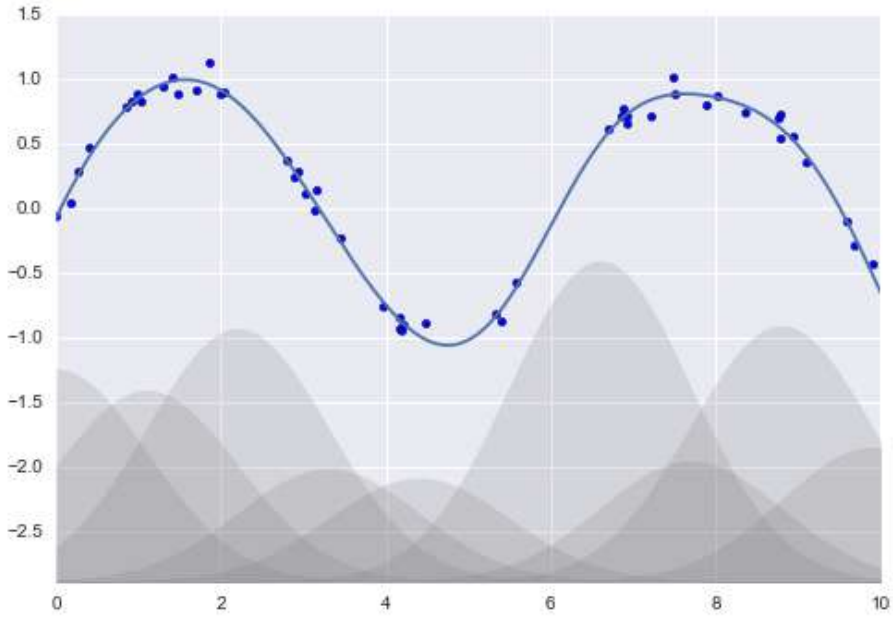
```
from sklearn.pipeline import make_pipeline
gauss_model = make_pipeline(GaussianFeatures(10, 1.0),
                            LinearRegression())
```

10 basic functions are needed!

```
gauss_model.fit(x[:, np.newaxis], y)
yfit = gauss _model.predict(xfit[:, np.newaxis])
```

```
fig, ax = plt.subplots()
ax.scatter(x, y)
ax.plot(xfit, gauss_model.predict(xfit[:, np.newaxis]))
```

Page 395



Fitting a Trigonometric function

- First with Sigmoidal basis functions

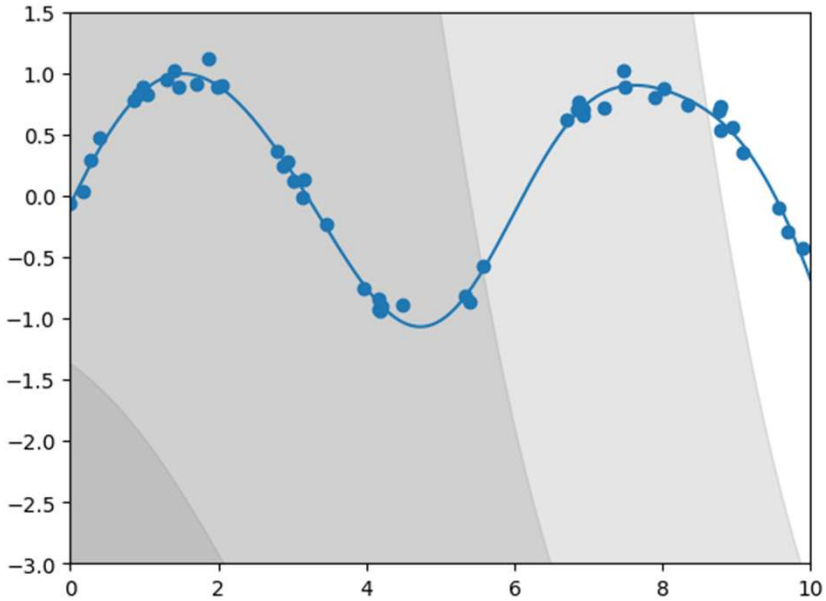
```
rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = np.sin(x) + 0.1 * rng.randn(50)
```

```
from sklearn.pipeline import make_pipeline
sigmoid_model = make_pipeline(SigmoidalFeatures(10, 1.0),
                              LinearRegression())
```

Used again 10 basic functions

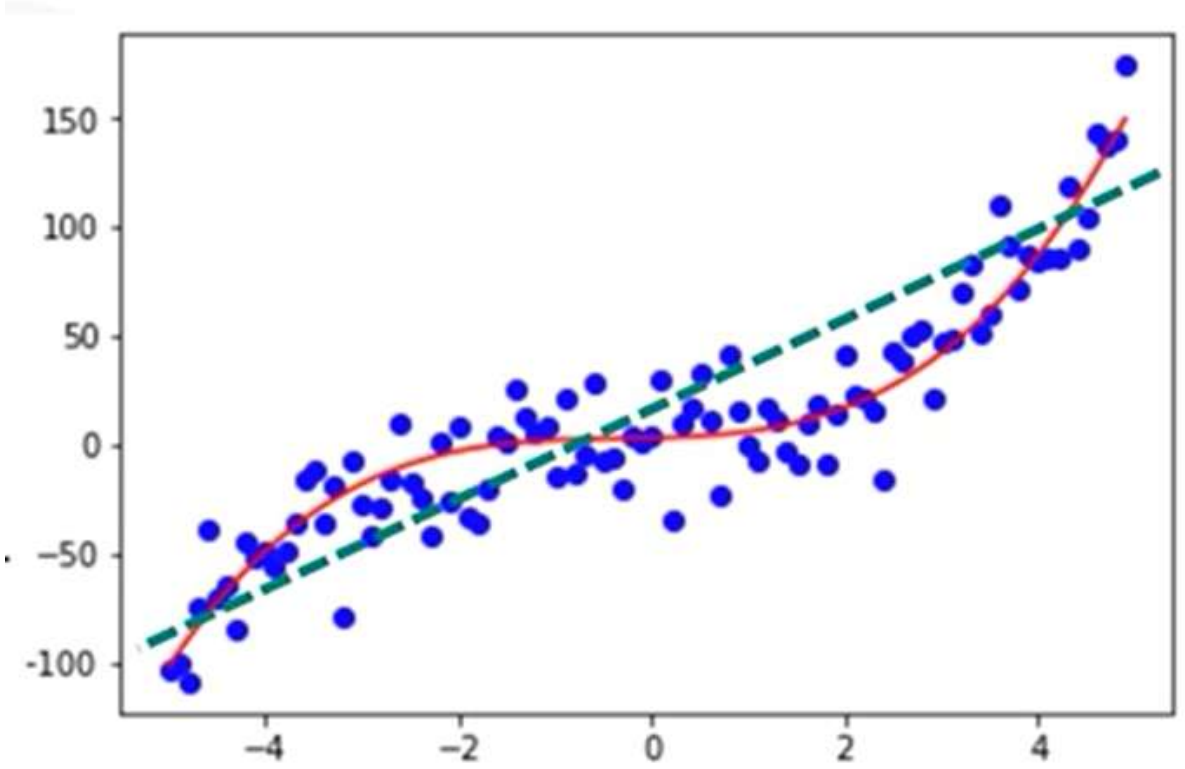
```
sigmoid_model.fit(x[:, np.newaxis], y)
yfit = sigmoid_model.predict(xfit[:, np.newaxis])
```

```
fig, ax = plt.subplots()
ax.scatter(x, y)
ax.plot(xfit, sigmoid _model.predict(xfit[:, np.newaxis]))
```



Polynomial regression

- Data is often non-linear, but shows a background pattern

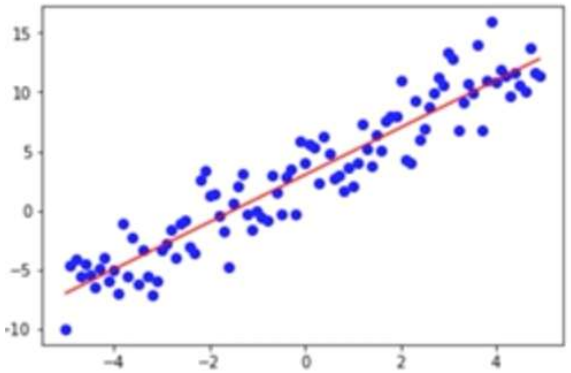


- A straight line “under-fits” the data
- The model is not complex enough

Slide idea from Benjamin Bengfort

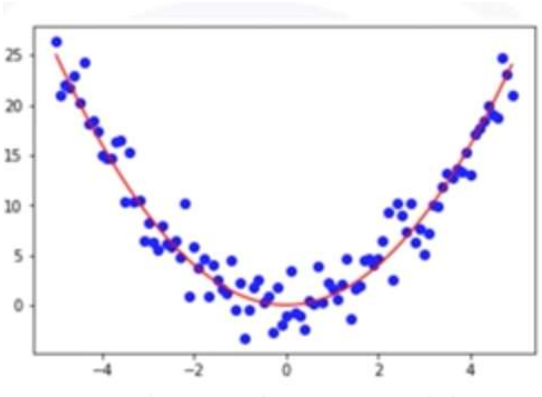
Polynomial regression

- You have to recognize the underlying trends



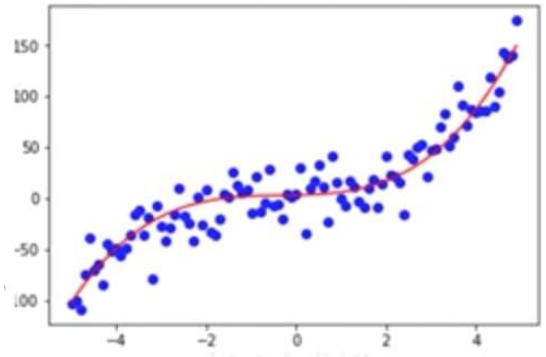
Linear

$y = \alpha + \beta x$



Quadratic

$y = \alpha + \gamma x^2$



Cubic

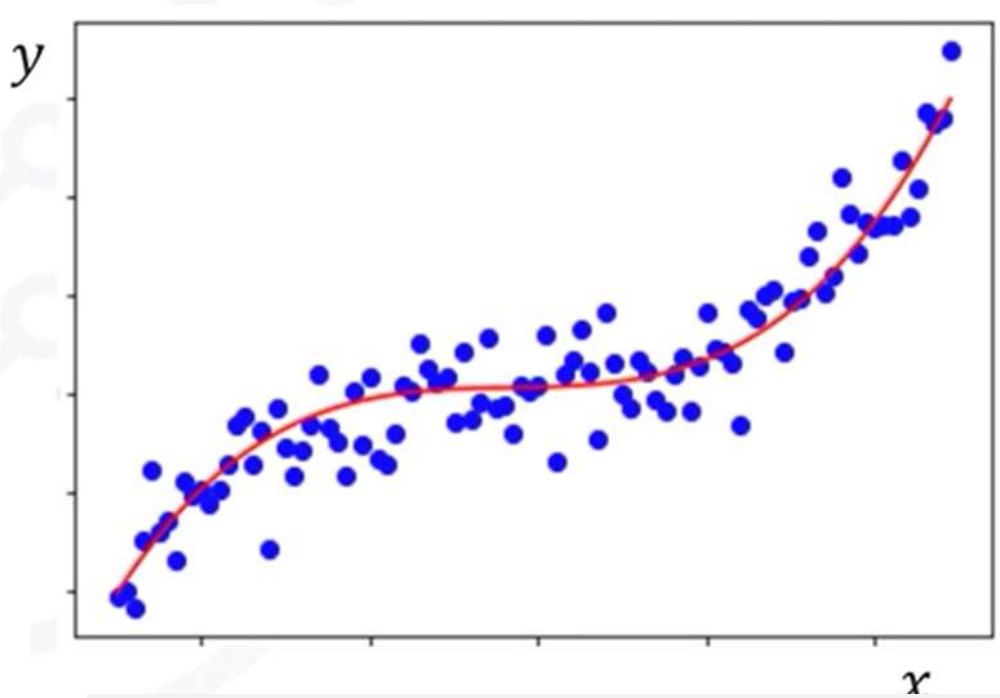
$y = \alpha + \delta x^3$

Slide idea from Benjamin Bengfort

Polynomial regression

- You recognized the cubic trend

$$y = \alpha + \beta x + \gamma x^2 + \delta x^3$$



Linearize

$$y = \alpha + \beta x + \gamma x' + \delta x''$$

with

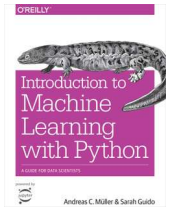
$$x' \equiv x^2$$

$$x'' \equiv x^3$$

```
poly = PolynomialFeatures(degree=3)
poly.fit(X_train)

X_poly = poly.transform(X_train)
```

```
reg = LinearRegression().fit(X_poly, y_train)
```

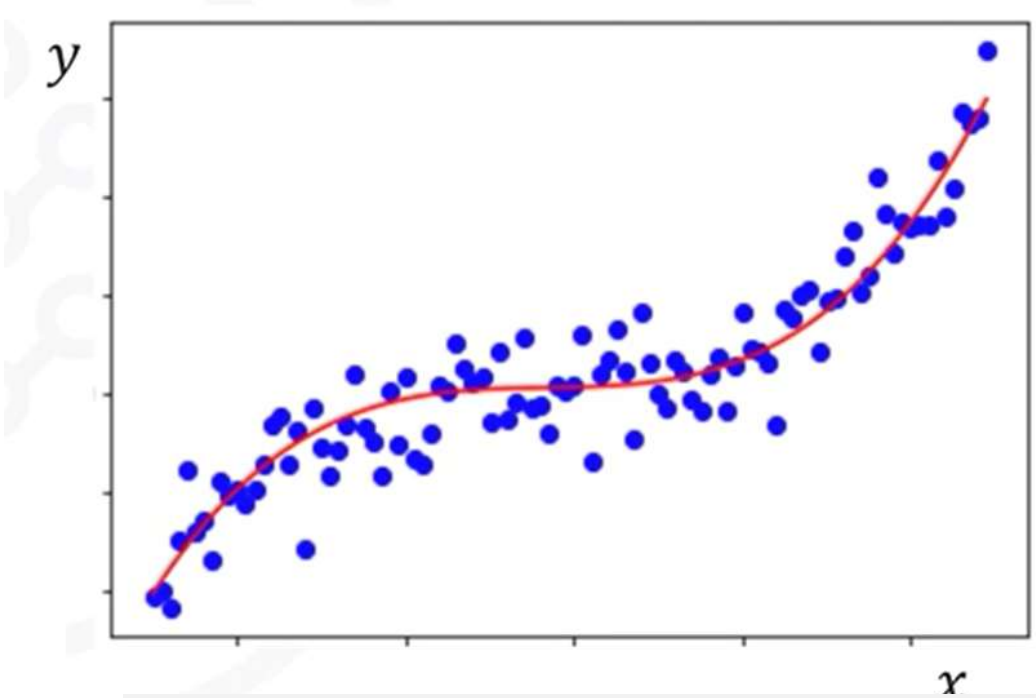
Slide idea from Benjamin Bengfort



Adding interaction features

- Expanding the feature space

$$y = \alpha + \beta x + \gamma x^2 + \delta x^3$$



Linearize

$$y = \alpha + \beta x + \gamma x' + \delta x''$$

with

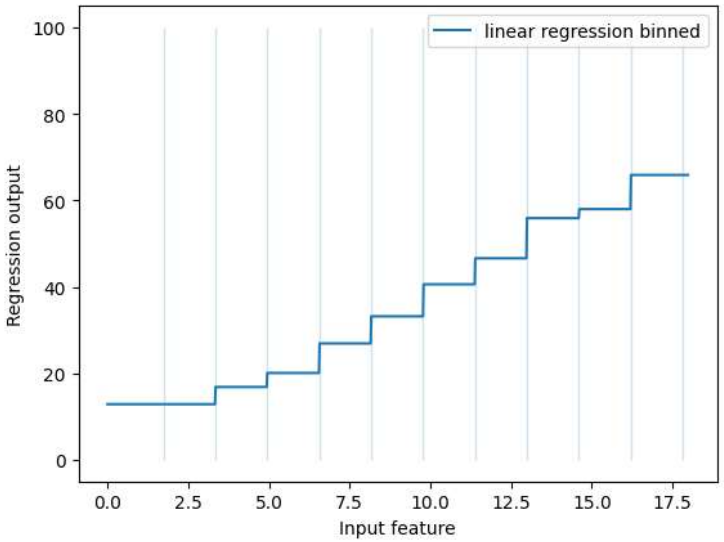
$$x' \equiv x^2$$

$$x'' \equiv x^3$$

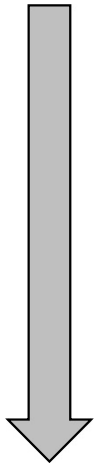
```
poly = PolynomialFeatures(degree=3)
poly.fit(X_train)

X_poly = poly.transform(X_train)          reg = LinearRegression().fit(X_poly, y_train)
```

Adding interaction features

- Expanding the feature space



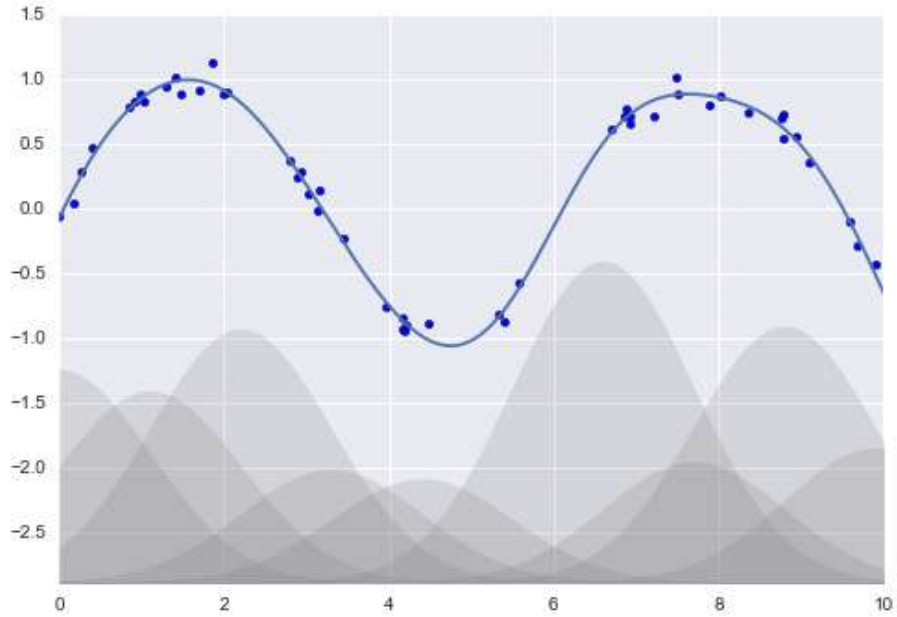
Discretize



```
kb = KBinsDiscretizer(n_bins=10, strategy='u
kb.fit(X_train)

X_binned = kb.transform(X_train)          reg = LinearRegression().fit(X_binned, y_train
```

Adding interaction features

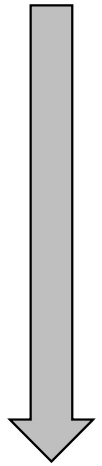
- Expanding the feature space



Basic functions

$$f(x) = \sum_{n=0}^{\infty} a_n (x - x_0)^n$$

$$\phi_n(x) = e^{-\frac{(x-\mu_n)^2}{2\sigma^2}}$$



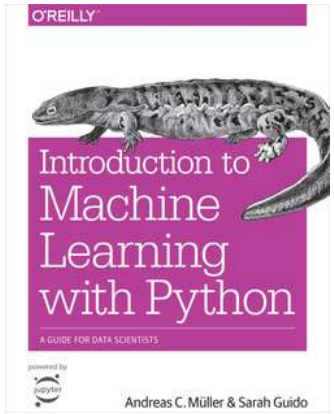
```
gauss_model = make_pipeline(GaussianFeatures(10, 1.0),
                            LinearRegression())
```

```
gauss_model.fit(x[:, np.newaxis], y)
```

Introduction to Machine Learning

- 4.5 Polynomial linear regression

- Strengths:
  - Easy to understand
  - Reflects underlying trend
- Weakness:
  - Extreme behavior at the edges



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

Conclusion

Learning outcomes of this course covered today

- Making the model more complex to prevent ‘under-fitting’