4. Stepwise Regression

Bruce E. Shapiro

Getting Started in Machine Learning

Last revised: February 3, 2019

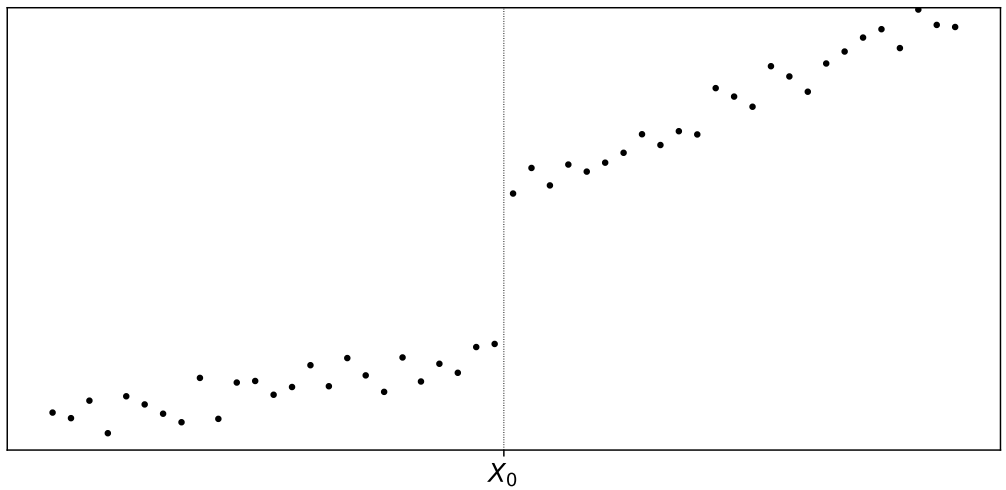
Goal of Stepwise (Piecewise) Regression

Given n points:

$$(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$$

fit a piecewise linear model, such as

$$f(x) = \begin{cases} a_0 + b_0 x, & x < X_0 \\ a_1 + b_1 x, & x \geq X_0 \end{cases}$$



More General Stepwise Problem

- Multiple linear segments

$$f(x) = \begin{cases} a_0 + b_0 x, & x < X_0 \\ a_1 + b_1 x, & x_0 \le x < X_1 \\ a_2 + b_2 x, & x_1 \le x < X_2 \\ \vdots & \\ a_n + b_n x, & x_{n-1} \le x < X_n \\ a_{n+1} + b_{n+1} x, & x \ge X_n \end{cases}$$

- The segments *may* be constrained to be continuous
- More generally, some of the segments may be nonlinear
- When the location of the steps is unknown, even the piecewise linear problem is nonlinear (finding the locations of the steps)

Constrained Stepwise Regression

- Two segments, constrain the fit to pass through the point (X_0, y_0).

Constrained Stepwise Regression

- Two segments, constrain the fit to pass through the point (X_0, y_0).
- Use the point-slope form on either side:

$$f(x) = \begin{cases} y_0 + b_0(x - X_0), & x < X_0 \\ y_0 + b_1(x - X_0), & x \geq X_0 \end{cases}$$

Constrained Stepwise Regression

- Two segments, constrain the fit to pass through the point (X_0, y_0).
- Use the point-slope form on either side:

$$f(x) = \begin{cases} y_0 + b_0(x - X_0), & x < X_0 \\ y_0 + b_1(x - X_0), & x \geq X_0 \end{cases}$$

- Objective function is nonlinear in unknown X_0:

$$\begin{aligned} \mathcal{E} &= \sum_{i=0}^{n-1} (f(x_i) - y_i)^2 \\ &= \sum_{x_i < X_0} (y_0 + b_0(x_i - X_0) - y_i)^2 + \sum_{x_i \geq X_0} (y_0 + b_1(x_i - X_0) - y_i)^2 \end{aligned}$$

Understanding Python Lambda Functions

- Lambda functions are **anonymous functions**, that is, functions without names

```
(lambda x:x**2)(7)
```

```
49
```

Understanding Python Lambda Functions

- Lambda functions are **anonymous functions**, that is, functions without names

```
(lambda x:x**2)(7)
```

```
49
```

```
(lambda x,y:x%y)(99,30)
```

```
9
```

Understanding Python Lambda Functions

- Lambda functions are **anonymous functions**, that is, functions without names

```
(lambda x:x**2)(7)
```

```
49
```

```
(lambda x,y:x%y)(99,30)
```

```
9
```

- lambda functions can be used anywhere regular functions are normally used

```
list(map(lambda x:x**2,range(5)))
```

```
[0, 1, 4, 9, 16]
```

numpy piecewise functions

- Used to define piecewise functions; returns array of values

```
piecewise(x, condition-list, function-list)
```

numpy piecewise functions

- Used to define piecewise functions; returns array of values

```
piecewise(x, condition-list, function-list)
```

- Example: Collatz function: $f(n) = \begin{cases} n/2, & \text{postive even} \\ 3n+1, & \text{positive odd} \end{cases}$

```
import numpy as np
def collatz(n):
    return (np.piecewise(n,
                         [n%2==0,n%2==1],
                         [lambda n:n/2, lambda n: 3*n+1]))
collatz(np.arange(1,100))
```

```
array([  4,   1,  10,   2,  16,   3,  22,   4,  28,
5,  34,   6,  40, 7,  46,   8,  52,   9,  58,  10,  64,
11,  70,  12,  76,  13, 82,  14,  88,  15,  94,  16, 100,
17, 106,  18, 112,  19, 118, 20, 124,  21, 130,  22, 136,
23, 142,  24, 148,  25, 154,  26, 160,  27, 166,  28, 172,
29, 178,  30, 184,  31, 190,  32, 196, 33, 202,  34, 208,
```

Optimize

Implement the function $f(x) = \begin{cases} y_0 + b_0(x - X_0), & x < X_0 \\ y_0 + b_1(x - X_0), & x \geq X_0 \end{cases}$

Optimize

Implement the function $f(x) = \begin{cases} y_0 + b_0(x - X_0), & x < X_0 \\ y_0 + b_1(x - X_0), & x \geq X_0 \end{cases}$

```
def f(x, X0, y0, b0,b1):
    return np.piecewise(x,
            [x < X0], [lambda x:y0+b0*(x-X0),
                       lambda x:y0+b1*(x-X0)])
```

Optimize

Implement the function $f(x) = \begin{cases} y_0 + b_0(x - X_0), & x < X_0 \\ y_0 + b_1(x - X_0), & x \geq X_0 \end{cases}$

```
def f(x, X0, y0, b0,b1):
    return np.piecewise(x,
            [x < X0], [lambda x:y0+b0*(x-X0),
                       lambda x:y0+b1*(x-X0)])
```

Solve for the coefficients:

```
optimize.curve_fit(f,X,Y,firstguess)
```

- Input X,Y are are X and Y data arrays
- Input first guess of parameters to f
- Output is list of parameters of f (e.g., X0, y0, b0,b1) and covariance matrix

Optimize

```
from scipy import optimize
parms, covmat = optimize.curve_fit(f,X,Y,[1,1,1,-1])
print(parms)
```

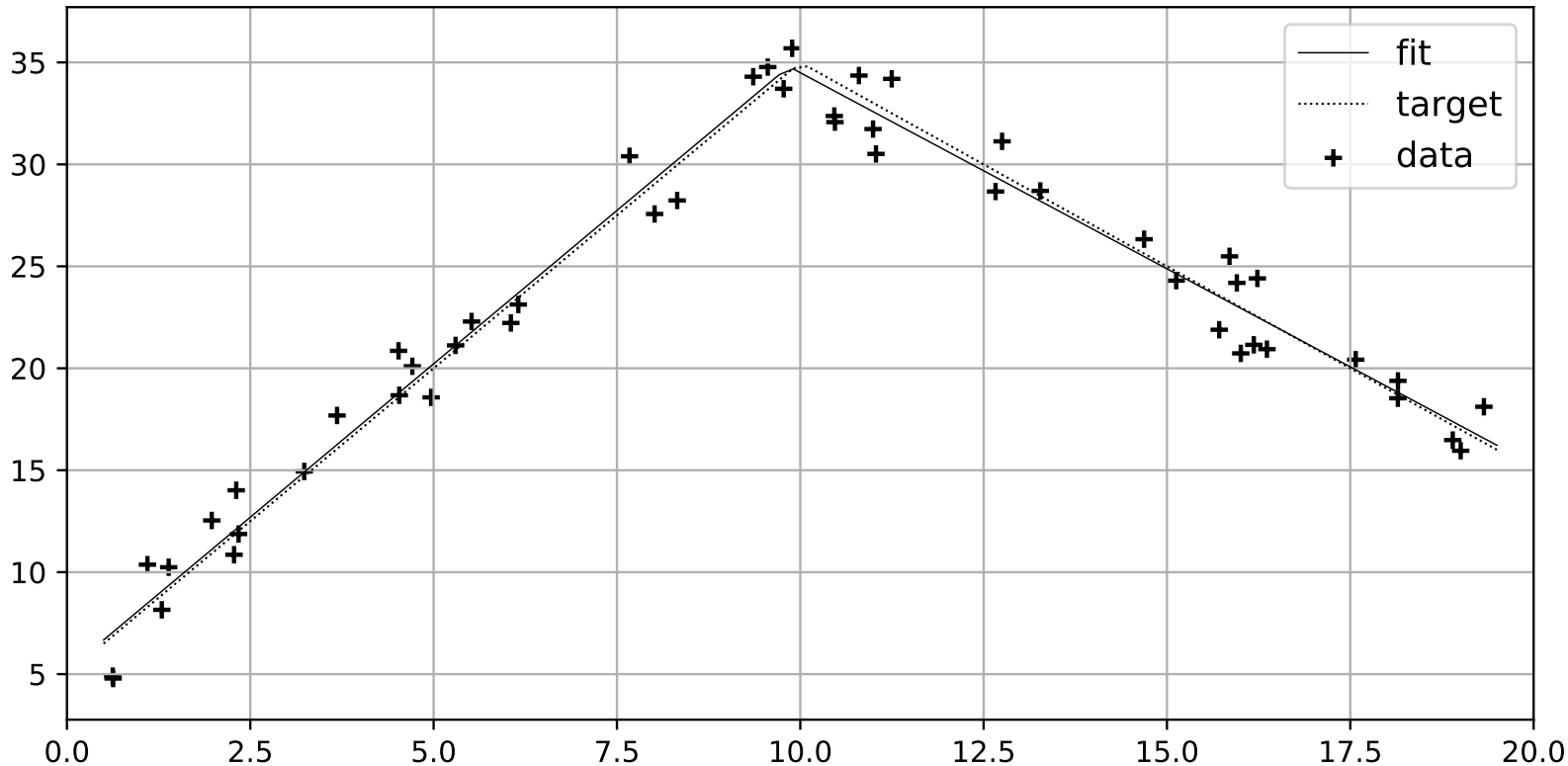
```
[ 9.8439359  34.79551181  3.00921039 -1.92330673]
```

Optimize

```
from scipy import optimize
parms, covmat = optimize.curve_fit(f,X,Y,[1,1,1,-1])
print(parms)
```

```
[ 9.8439359  34.79551181  3.00921039 -1.92330673]
```

```
plt.plot(xline,[f(x,*parms) for x in xline],c="k",
   label="fit",lw=.5)
plt.plot(xline,yline,c="k",ls=":",label="target",lw=.75)
plt.scatter(xran,yran, marker="+",c="k",label="data")
plt.legend(fontsize=12)
plt.grid()
plt.xlim(0,20)
```

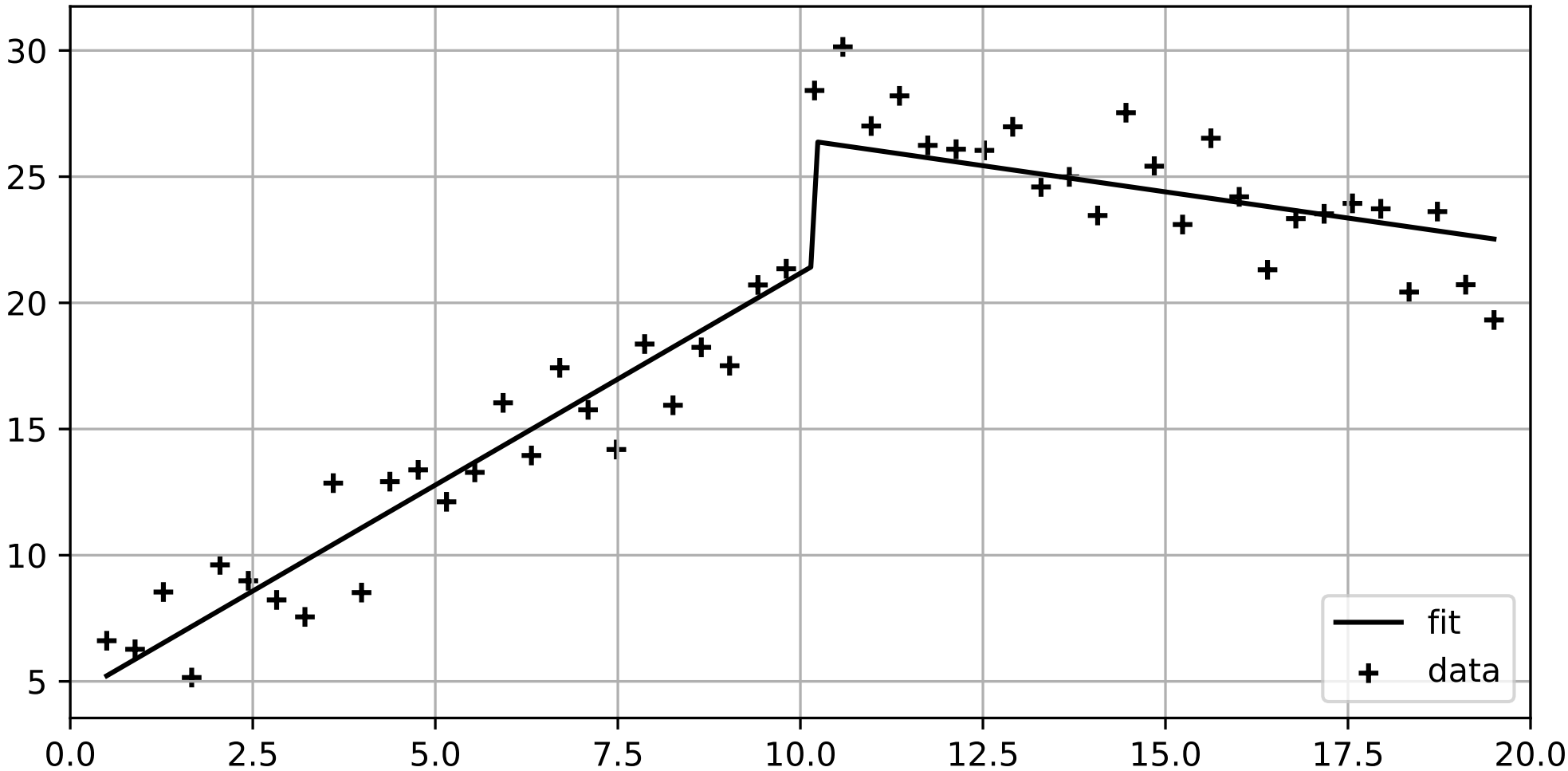

fit
target
data
35
30
25
20
15
10
5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Unconstrained Fit

```
def f(x, x0, y0, y1, b0, b1):
    return np.piecewise(x,
            [x < x0], [lambda x:y0+b0*(x-x0),
                       lambda x:y1+b1*(x-x0)])
#
# ... not shown ... code to generate toy data (X, Y)
#
parms=optimize.curve_fit(f, X, Y, [8,15,25,1,-1])[0];
parms
```

```
array([10.19387744, 21.5015197 , 26.39042311,
  1.67933918, -0.41477884])
```

```
xline=np.linspace(0.5,19.5,200)
plt.plot(xline,[float(f(u,*parms)) for u in xline],
         label="fit",c="k")
```


30
25
20
15
10
5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
fit
data

Citations

1