CME 252: Model Fitting

AJ Friend
ICME, Stanford University

Linear least squares

Outline

Linear least squares overview

- Ubiquitous statistical model
- Applications everywhere
- Goal: find linear model $f(x)$ that fits your data
- Useful model for the purpose of studying optimization

Let's start with data

x: independent variable	y: response variable
0.0	0.46
0.11	0.31
0.22	0.38
0.33	0.39
0.44	0.65
0.56	0.4
0.67	0.87
0.78	0.69
0.89	0.87
1.0	0.88

Where might this data come from?

x: independent variable	y: response variable
height	weight
square feet	price of home
device property	failure rate
stock market return	individual asset return

Where did this data come from?

A linear model with random error:

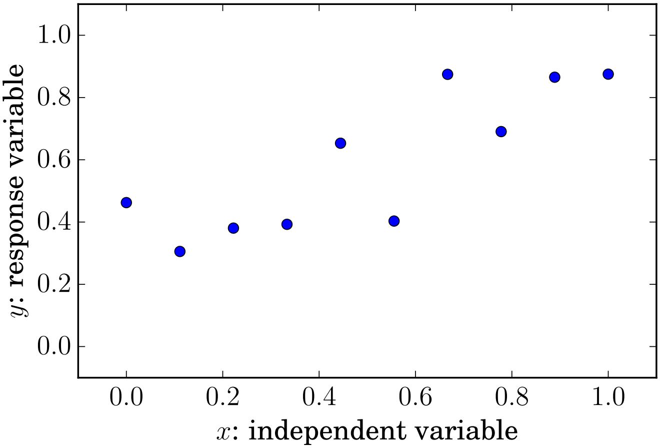
$$y_i = m \cdot x_i + b + \epsilon_i$$

The error is standard normal: $\epsilon_i \sim N(0, \sigma)$

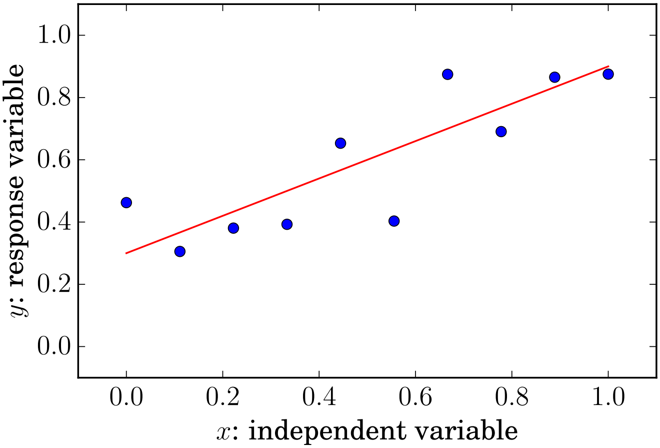
Code to generate in python:

```
np.random.seed(1)
m = 0.6
b = 0.3
sigma = .1
x = np.linspace(0,1,10)
y = m*x + b + sigma*np.random.standard_normal(x.shape)
```

Let's plot the data



Let's draw a line through it



Why do we want to do this?

- We have data. In the previous figures we show 2-dimensional data with points (x_i, y_i)
- Want to better understand data
- Want to use data for useful things, for example to make predictions
- We can do both by building a model

$$y \approx f(x)$$

- Model has parameters. We use optimization to compute those parameters.

Linear models in one dimension

The model is:

$$y \approx m \cdot x + b$$

- x is the independent variable
- y is the dependent or response variable
- m is the slope
- b is the y-intercept
- This is linear regression

Fitting the model to data

- Any given data point is going to result in some model error
- In optimization and linear algebra, we call this error the *residual*:

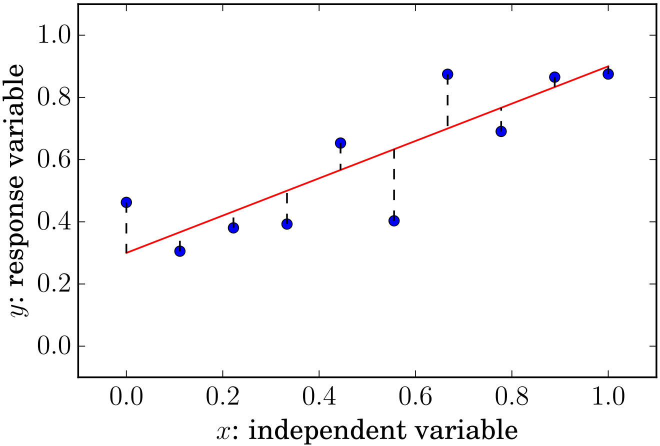
$$r_i = m \cdot x_i + b - y_i$$

- To fit the model to data, we set up an optimization problem that chooses parameters m and b to minimize the sum of squared residuals:

$$\text{minimize} \quad \tfrac{1}{2} \textstyle\sum_{i=1}^{n} (m \cdot x_i + b - y_i)^2$$

- If the errors are distributed according to the normal distribution, then the solution to this optimization problem maximizes the log-likelihood of the model.

Fitting the model to data



Solve in CVXPY

Remember the optimization problem: minimize $\frac{1}{2}\sum_{i=1}^{n}(m \cdot x_i + b - y_i)^2$

We can write this almost directly in python:

```
m = Variable()
b = Variable()
objective = Minimize(sum_squares(m*x + b - y))
prob = Problem(objective)
result = prob.solve()
```

```
m.value = 0.56604, b.value = 0.30727
```

Loss functions

- sum-of-squares is a **loss function** describing how unhappy we are with misfit of model
$$y \approx m \cdot x + b$$
- loss function makes "$\approx$" precise
- sum-of-squares is not the only choice!

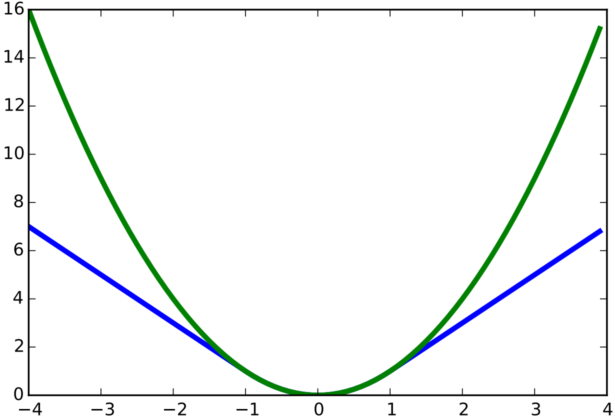
Huber function

- huber function allows us to better handle **outliers**
- defined as

$$h_M(x) = \begin{cases} x^2 & |x| \leq M \\ 2M|x| - M^2 & |x| > M \end{cases}$$

- usual quadratic loss in interval $[-M, M]$
- linear loss for $|x| > M$
- linear penalty for large errors is much less severe than quadratic
- large errors are better 'tolerated', have less influence on fit

Huber function



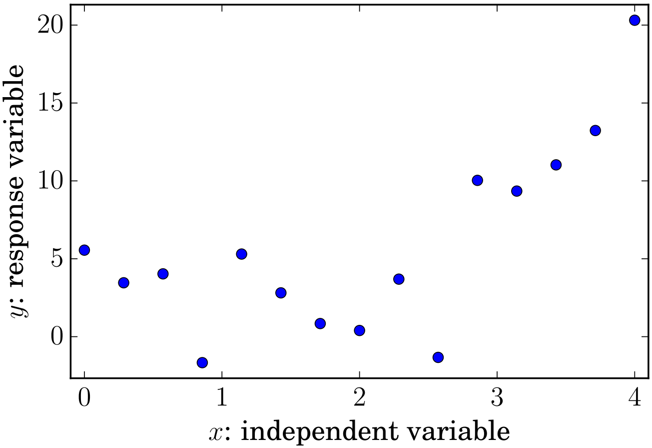
Huber notebook

notebook

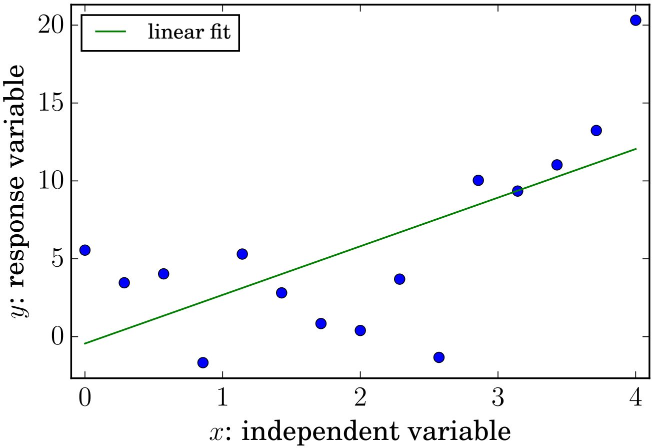
Non-linear data

Outline

What about this data?



Let's fit a linear model



We can fit an exponential model

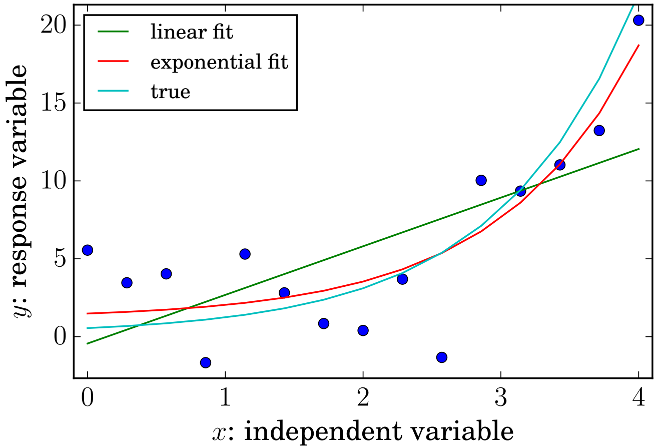
The model:

$$y \approx m \cdot e^x + b$$

Note: this model is still linear in the parameters m and b! We just need to transform the independent variable and then solve using the same technique. In code with CVXPY:

```
m = Variable()
b = Variable()
objective = Minimize(sum_squares(m*np.exp(x) + b - y))
prob = Problem(objective)
result = prob.solve()
```

Result



linear fit
exponential fit
true
20
15
10
5
0
0
1
2
3
4
y: response variable
x: independent variable

Least squares in matrix-vector form

Outline

More variables

- We've been talking about least squares with one independent variable
- Utility of linear models is increased when we incorporate more variables
- For this discussion, we need vectors and matrices!

Least squares objective

- Let's say we have a data set (a_i, y_i)
- a is now the independent variable and y is the response variable
- The linear model: $y = m \cdot a + b + r$
- The optimization formulation to find parameters m and b is

$$\text{minimize} \quad \frac{1}{2}\sum_{i=1}^{n}(m \cdot a_i + b - y_i)^2 = \frac{1}{2}||m \cdot a + b - y||_2^2 = \frac{1}{2}||r||_2^2$$

Matrix-vector form for least squares

$$\text{minimize} \quad \tfrac{1}{2}\textstyle\sum_{i=1}^{n}(m \cdot a_i + b - y_i)^2$$

- Pack data from the independent variable and a constant into matrix A and model parameters into vector x:

$$A = \begin{pmatrix} a_1 & 1 \\ a_2 & 1 \\ \vdots & \vdots \\ a_n & 1 \end{pmatrix}, \qquad x = \begin{bmatrix} m \\ b \end{bmatrix}$$

- Linear model for the data is now:

$$y = Ax + r$$

Standard form for least squares

$$\text{minimize} \quad \frac{1}{2}||Ax - b||_2^2$$

In the context of model fitting:

- A is a matrix that contains data from independent variables
- b is the vector holding response data
- x is the vector of model parameters
- The optimization problem above is solved to find x^*, the parameters that minimize the sum of squared residuals
- For each item of data, we have the equation where a_i^T is row i of A

$$a_i^T x - b_i = r_i$$

Notation from statistics

$$\text{minimize} \quad \frac{1}{2}||\mathbf{y} - \mathbf{X}\beta||_2^2$$

The statistics community often uses different notation:

- $\mathbf{X}$ is the matrix of input data
- $\mathbf{y}$ is the vector of response data
- β is the vector of model parameters

CVXPY for least squares

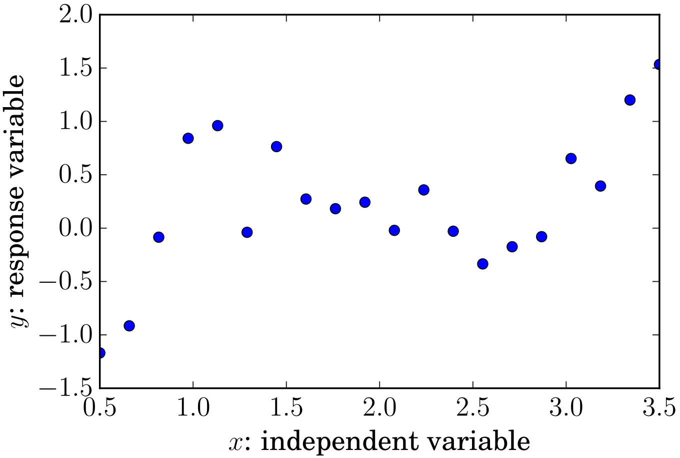
```
x = Variable(A.shape[1])
objective = Minimize(sum_squares(A*x - b))
prob = Problem(objective)
prob.solve()
```

```
slope = 0.566, intercept = 0.3073
```

Examples

Outline

What about this data?



We can try a polynomial model

- Have m data points (u_i, y_i)
- The model

$$y \approx p(u) = x_1 + x_2 u + x_3 u^2 + \cdots + x_n u^{n-1}$$

- u is the independent variable
- y is the response variable
- x_i are the model parameters and coefficients of the polynomial
- Model is linear in the parameters! We can use least squares!

Linear model

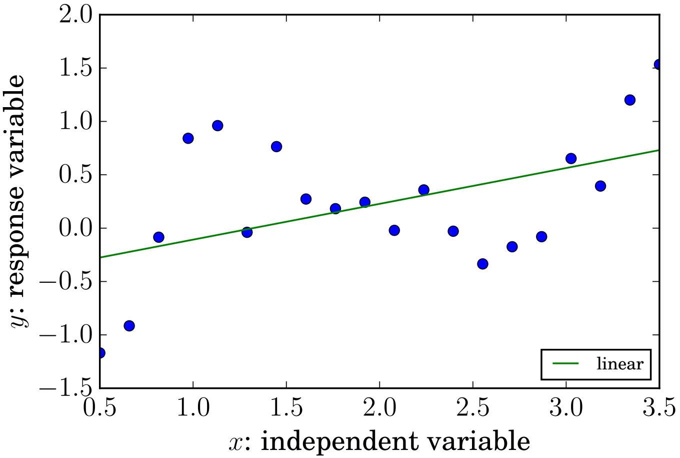
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_m \end{bmatrix} \approx \begin{bmatrix} 1 & u_1 & u_1^2 & \dots & u_1^{n-1} \\ 1 & u_2 & u_2^2 & \dots & u_2^{n-1} \\ 1 & u_3 & u_3^2 & \dots & u_3^{n-1} \\ 1 & u_4 & u_4^2 & \dots & u_4^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & u_m & u_m^2 & \dots & u_m^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

$$y \approx Ax$$

Solve with CVXPY

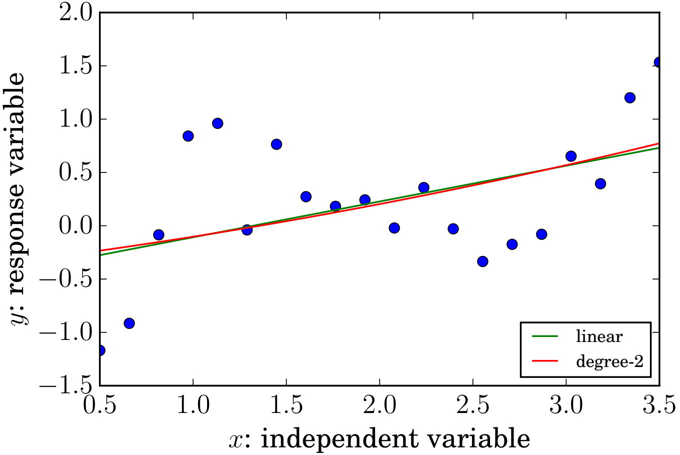
```
def cvxpy_poly_fit(x,y,degree):
    # construct data matrix
    A = np.vander(x,degree+1)
    b = y
    p_cvx = Variable(degree+1)
    # set up optimization problem
    objective = Minimize(sum_squares(A*p_cvx - b))
    constraints = []
    # solve the problem
    prob = Problem(objective,constraints)
    prob.solve()
    # return the polynomial coefficients
    return np.array(p_cvx.value)
```

Linear fit

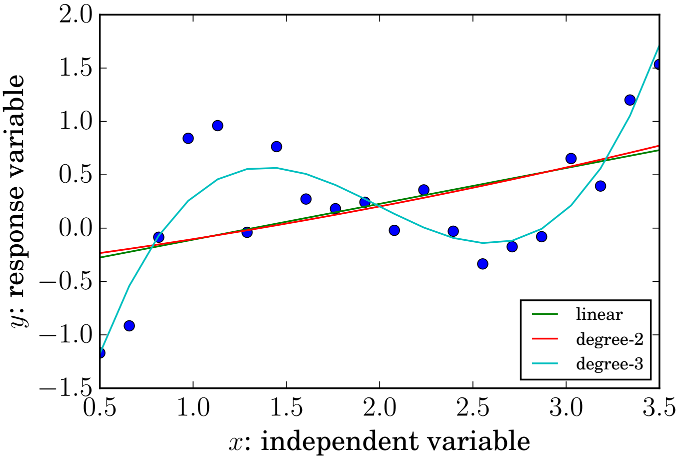


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.5
1.0
1.5
2.0
2.5
3.0
3.5
y: response variable
x: independent variable
linear

Quadratic fit



Cubic fit



2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.5
1.0
1.5
2.0
2.5
3.0
3.5
y: response variable
x: independent variable
linear
degree-2
degree-3

Generating model