Lecture 2: Reviewing the basics of machine learning

Announcements:

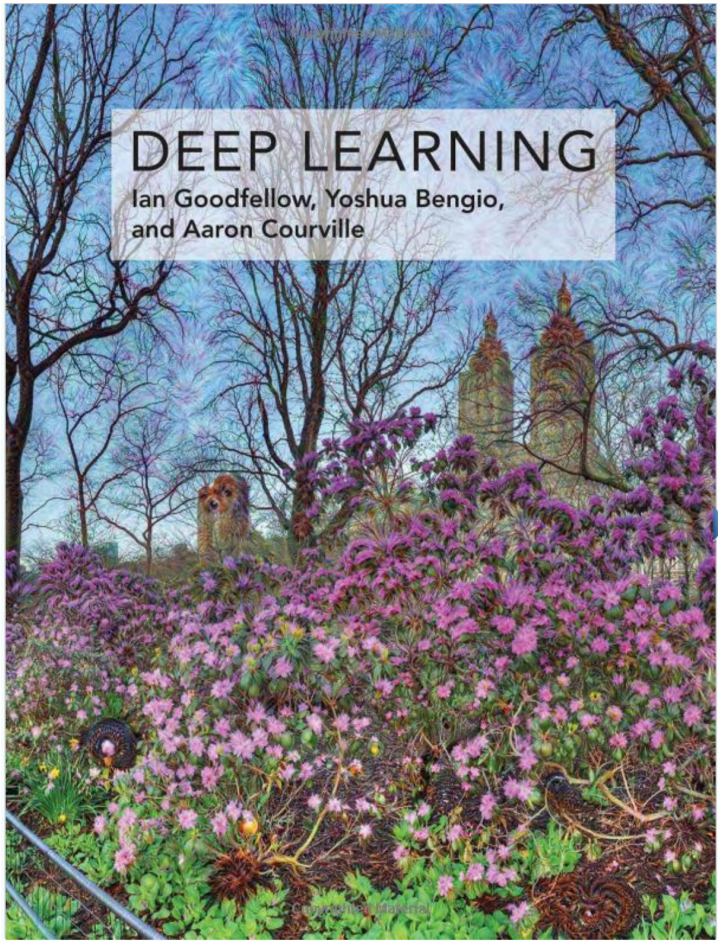
- We sent out an Announcement on Bruin Learn on how to sign up for Gradescope and Piazza, as well as a poll on discussion section scheduling.
- We are still setting the discussion sections and OH based on the polls. We will send a Bruin Learn announcement when these are set (likely this evening; tomorrow at the latest).
- We uploaded a few supporting files to Bruin Learn: a pdf on setting up Python and Jupyter Notebooks, and a pdf on using Google Colab (optional). You are welcome to use whatever setup you desire. I would personally install anaconda and create conda virtual environments.
- We also uploaded the formal notes of the class and exams dating back to 2018.
- Any questions on the syllabus or logistics?

Lecture 2: Basics of machine learning

Reading:

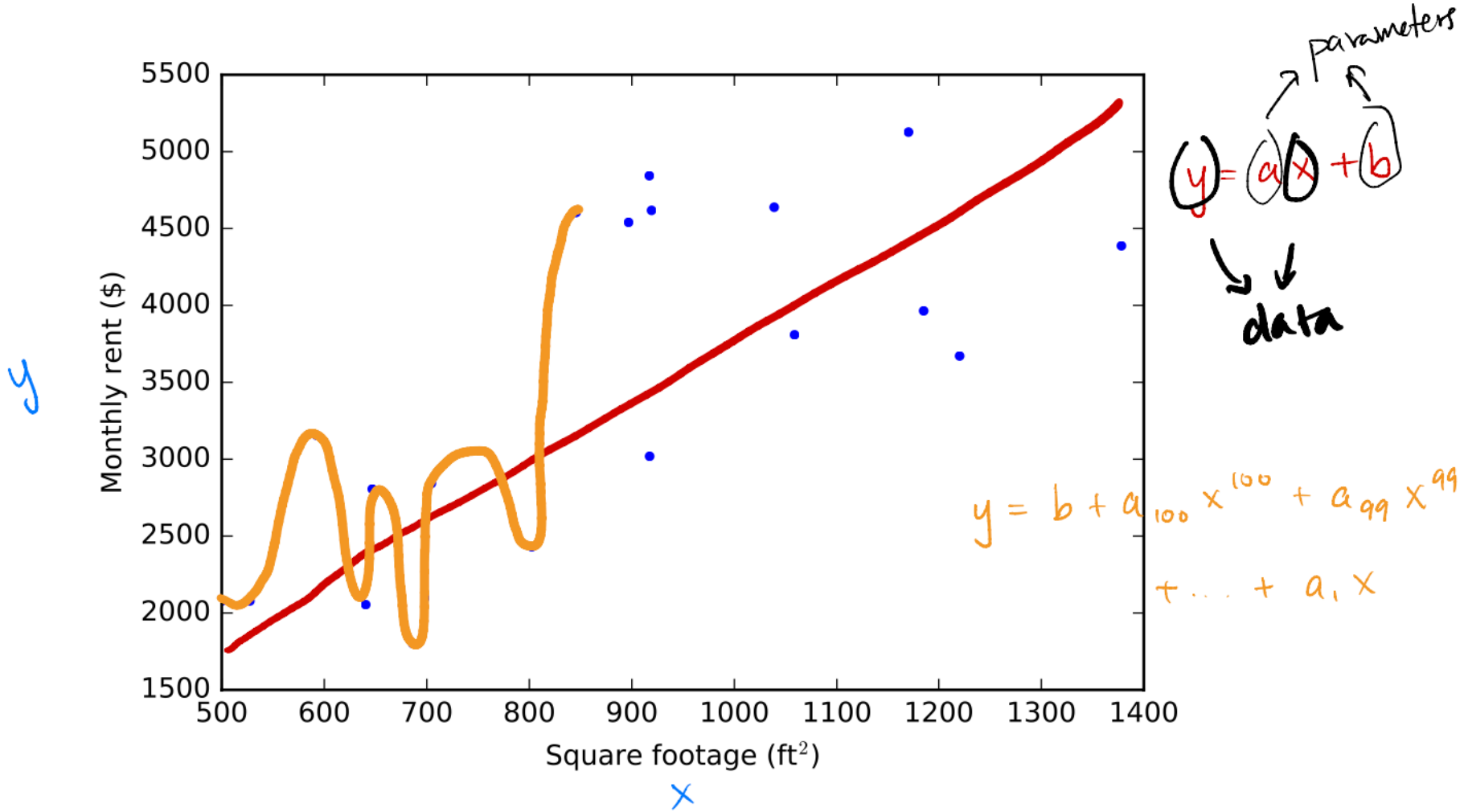
Deep Learning, chapter 5 (up to and including section 5.5).

To refresh your linear algebra and probability, look at chapters 2 and 3 respectively.



An example of supervised learning

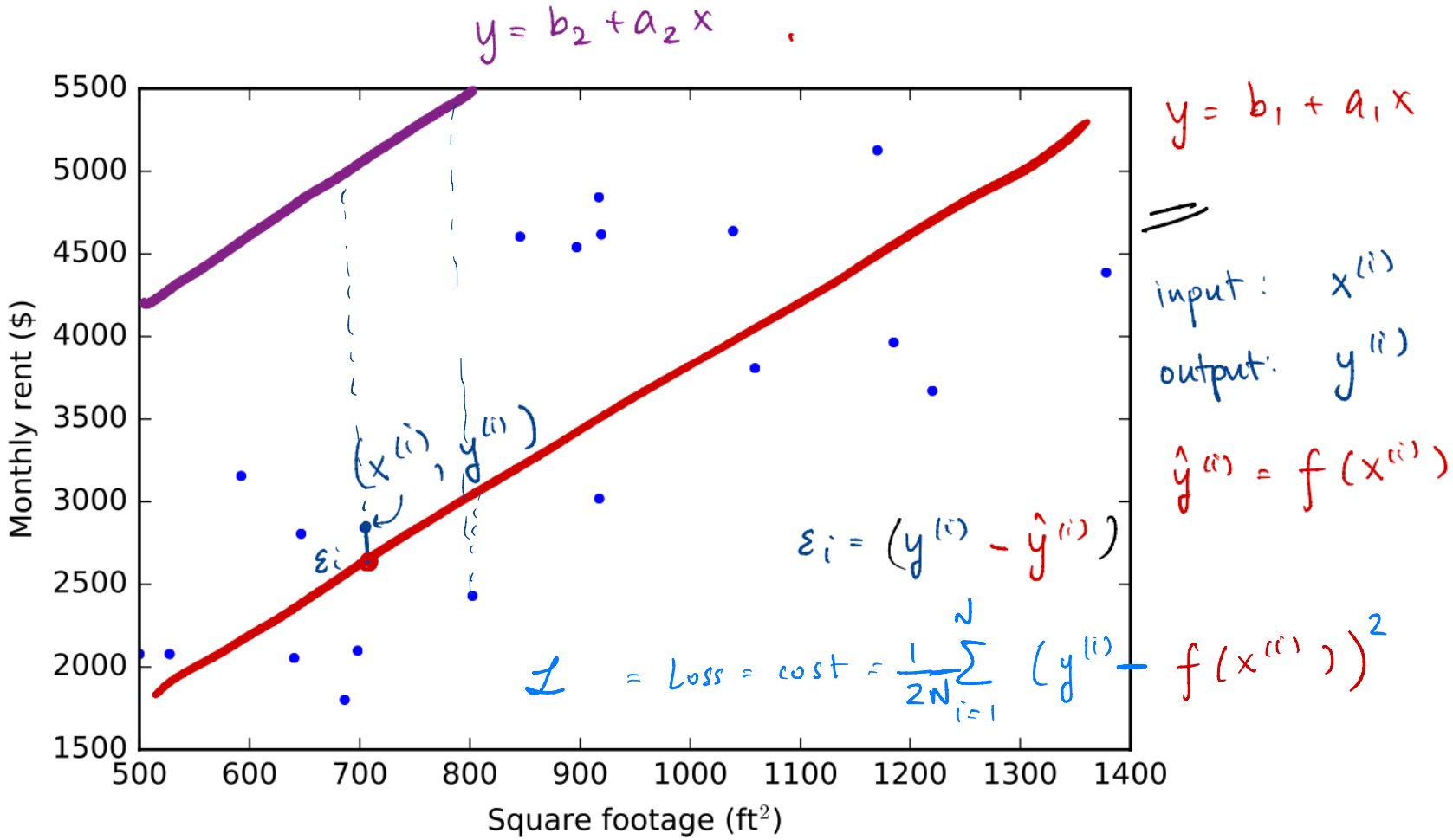
$f(x) = ax + b$



How should we model this data?

- Inputs, **x**? Outputs, **y**?
- What model should we use?
- How do we assess how good our model is?

An example of supervised learning



How should we model this data?

- Inputs, **x**? Outputs, **y**?
- What model should we use?
- How do we assess how good our model is?

An example of supervised learning

A simple linear example:

- Model:

$$\begin{aligned} \hat{y} &= ax + b \\ &= \theta^T \hat{\mathbf{x}} \end{aligned}$$

data given to us

output: $\hat{y}$

input: x

parameters: I get to choose the values of a, b to make the model as good as poss.

$\theta = \begin{bmatrix} a \\ b \end{bmatrix}$ $\hat{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$ to make $\mathcal{L}(a, b)$ as small as poss.

- Cost function:

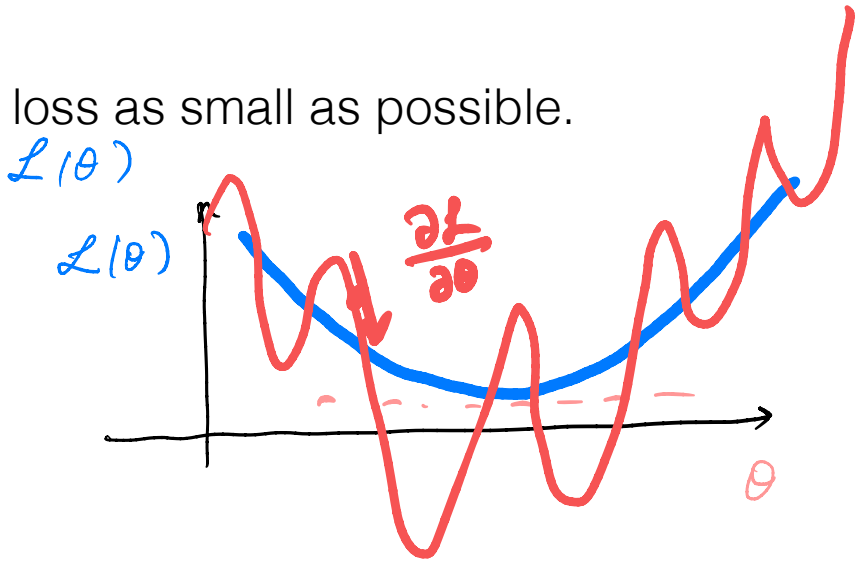
$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{2} \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2 \\ &= \frac{1}{2} \sum_{i=1}^{N} (y^{(i)} - \theta^T \hat{\mathbf{x}}^{(i)})^2 \end{aligned}$$

How do we learn the parameters of this model?

An example of supervised learning

Construct it as an optimization problem.

Our goal is to choose the parameters to make the loss as small as possible.



Thus our strategy to find the best θ is to:

- Calculate

$$\frac{d\mathcal{L}}{d\theta}$$

- Solve for θ such that

$$\frac{\partial \mathcal{L}}{\partial \theta} = 0$$

However, θ is a vector, so how do we take derivatives with respect to it?

Aside: vector and matrix derivatives

In machine learning, we often take derivatives with respect to vectors and matrices.

These are typically called *gradients*, and in this class we'll use the following notation to denote derivatives with respect to vectors and matrices.

In this class, we will use both the differentiation operator ∂ and ∇ to denote derivatives. If y is a scalar, and $\mathbf{x}$ is a vector, then the gradient of y with respect to $\mathbf{x}$ is denoted as both:

$$\frac{\partial y}{\partial \mathbf{x}} \quad \text{and} \quad \nabla_{\mathbf{x}} y$$

The gradient of y with respect to $\mathbf{x}$ is itself a vector with the same dimensionality as $\mathbf{x}$.

Aside: vector and matrix derivatives

y scalar

x vector $\in \mathbb{R}^n$ $\quad \frac{\partial y}{\partial x}$ or $\nabla_x y \in \mathbb{R}^n$

The gradient

The gradient generalizes the scalar derivative to multiple dimensions. Suppose $f : \mathbb{R}^n \to \mathbb{R}$ transforms a vector $\mathbf{x} \in \mathbb{R}^n$ to a scalar. If $y = f(\mathbf{x})$, then the gradient is:

$$\nabla_{\mathbf{x}} y = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix}$$

$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ $\quad \frac{\partial y}{\partial x} = \begin{bmatrix} 1 \\ 0.5 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ $\quad \frac{\partial y}{\partial x_1} = 1$

$\frac{\partial y}{\partial x_2} = 0.5$

Δy due to Δx_1 is $\approx \frac{\partial y}{\partial x_1} \cdot \Delta x_1$

Aside: vector and matrix derivatives

In other words, the gradient is:

- A *vector* that is the same size as $\mathbf{x}$, i.e., if $\mathbf{x} \in \mathbb{R}^n$ then $\nabla_{\mathbf{x}} y \in \mathbb{R}^n$.
- Each dimension of $\nabla_{\mathbf{x}} y$ tells us how small changes in $\mathbf{x}$ in that dimension affect y. i.e., changing the ith dimension of $\mathbf{x}$ by a small amount, Δx_i, will change y by

$$\frac{\partial y}{\partial x_i} \Delta x_i$$

We may also denote this as:

$$(\nabla_{\mathbf{x}} y)_i \, \Delta x_i$$

$\Delta \vec{x} = \begin{bmatrix} 0.05 \\ 0.01 \\ 0.02 \\ \vdots \end{bmatrix}$ → how much does y change?

$\nabla_x y$

$$\Delta y \approx \sum_{i=1}^{n} \frac{\partial y}{\partial x_i} \Delta x_i$$

$$\approx (\nabla_x y)^T \Delta x$$

Aside: vector and matrix derivatives

Example: derivative with respect to a vector

Let $f(\mathbf{x}) = \theta^T \mathbf{x}$. What is $\nabla_{\mathbf{x}} f(\mathbf{x})$?

$\theta, \vec{x} \in \mathbb{R}^n$

$y = f(x)$

$y \in \mathbb{R}$

$\nabla_x y \in \mathbb{R}^n$

$$y = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

$$\nabla_x y = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

$$\boxed{\nabla_x f(x) = \theta}$$

Aside: vector and matrix derivatives

Example: derivative with respect to a vector

Let $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A}\mathbf{x}$. What is $\nabla_{\mathbf{x}} f(\mathbf{x})$?

$x \in \mathbb{R}^n \qquad A \in \mathbb{R}^{n \times n}$

$\nabla_x f(x) \in \mathbb{R}^n$

$(1 \times n)\,(n \times n)\,(n \times 1)$

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \\ a_{n1} & & a_{nn} \end{bmatrix}$$

$$f(x) = x^T A x = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot x_i \cdot x_j$$

$$\frac{\partial f(x)}{\partial x_1} = 2a_{11} x_1 + \sum_{j=2}^{n} a_{1j} \cdot x_j + \sum_{i=2}^{n} a_{i1} x_i$$

$$= \sum_{j=1}^{n} a_{1j} \cdot x_j + \sum_{i=1}^{n} a_{i1} x_i$$

$i = 1, j = 1$

$$\frac{\partial\, a_{11} \cdot x_1^2}{\partial x_1} = 2a_{11} \cdot x_1$$

$i = 1, j \neq 1$

$$\frac{\partial \left(\sum_{j=2}^{n} a_{1j} \cdot x_1 \cdot x_j \right)}{\partial x_1}$$

$i \neq 1, j = 1$

$$\frac{\partial \sum_{i=2}^{n} a_{i1} \cdot x_i \cdot x_1}{\partial x_1}$$

Aside: vector and matrix derivatives

$$\frac{\partial f(x)}{\partial x_1} = \sum_{j=1}^{n} a_{1j} \cdot x_j + \sum_{i=1}^{n} a_{i1} x_i$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{n1} & & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

A $\qquad$ x

$$(Ax)_1 + (A^T x)_1$$

$$\begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} = \begin{bmatrix} (Ax)_1 + (A^T x)_1 \\ (Ax)_2 + (A^T x)_2 \\ \vdots \end{bmatrix}$$

$\frac{\partial f(x)}{\partial x_2}$, $\frac{\partial f(x)}{\partial x_3}$ all follow the same pattern

$$\nabla_x f(x) = \nabla_x (x^T A x) = (Ax + A^T x)$$

$$= (A + A^T) x$$

if A is symmetric

$$= 2Ax$$

when $n = 1$

$f(x) = x \cdot a \cdot x = a \cdot x^2$

$\frac{\partial f(x)}{\partial x} = 2ax$

Aside: vector and matrix derivatives

First, we note that $\mathbf{x}^T\mathbf{A}\mathbf{x} = \sum_i \sum_j a_{ij}x_i x_j$. Then, we have

$$\begin{aligned}
\nabla_{\mathbf{x}} f(\mathbf{x}) &= \begin{bmatrix} 2a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + a_{21}x_2 + \cdots + a_{n1}x_n \\ 2a_{22}x_2 + a_{21}x_1 + \cdots + a_{2n}x_n + a_{12}x_1 + \cdots + a_{n2}x_n \\ \vdots \\ 2a_{nn}x_n + a_{n1}x_1 + \cdots + a_{n,n-1}x_{n-1} + a_{1n}x_1 + \cdots + a_{n-1,n}x_{n-1} \end{bmatrix} \\
&= \mathbf{A}\mathbf{x} + \mathbf{A}^T\mathbf{x}
\end{aligned}$$

Matrix derivatives

$y = z^T A x$ $\quad z \in \mathbb{R}^m$ $\quad x \in \mathbb{R}^n$ $\quad A \in \mathbb{R}^{m \times n}$

$$\frac{\partial y}{\partial A} = \nabla_A y = \begin{bmatrix} \frac{\partial y}{\partial a_{11}} & \frac{\partial y}{\partial a_{12}} & \cdots & \frac{\partial y}{\partial a_{1n}} \\ \vdots & & & \vdots \\ \frac{\partial y}{\partial a_{m1}} & & \cdots & \frac{\partial y}{\partial a_{mn}} \end{bmatrix}$$

if $A \in \mathbb{R}^{m \times n}$

$\nabla_A y \in \mathbb{R}^{m \times n}$

"Denominator Layout"

$x \in \mathbb{R}^{n \times 1}$, $\nabla_x y \in \mathbb{R}^{n \times 1}$

$A \in \mathbb{R}^{m \times n}$, $\nabla_A y \in \mathbb{R}^{m \times n}$

"Numerator Layout"

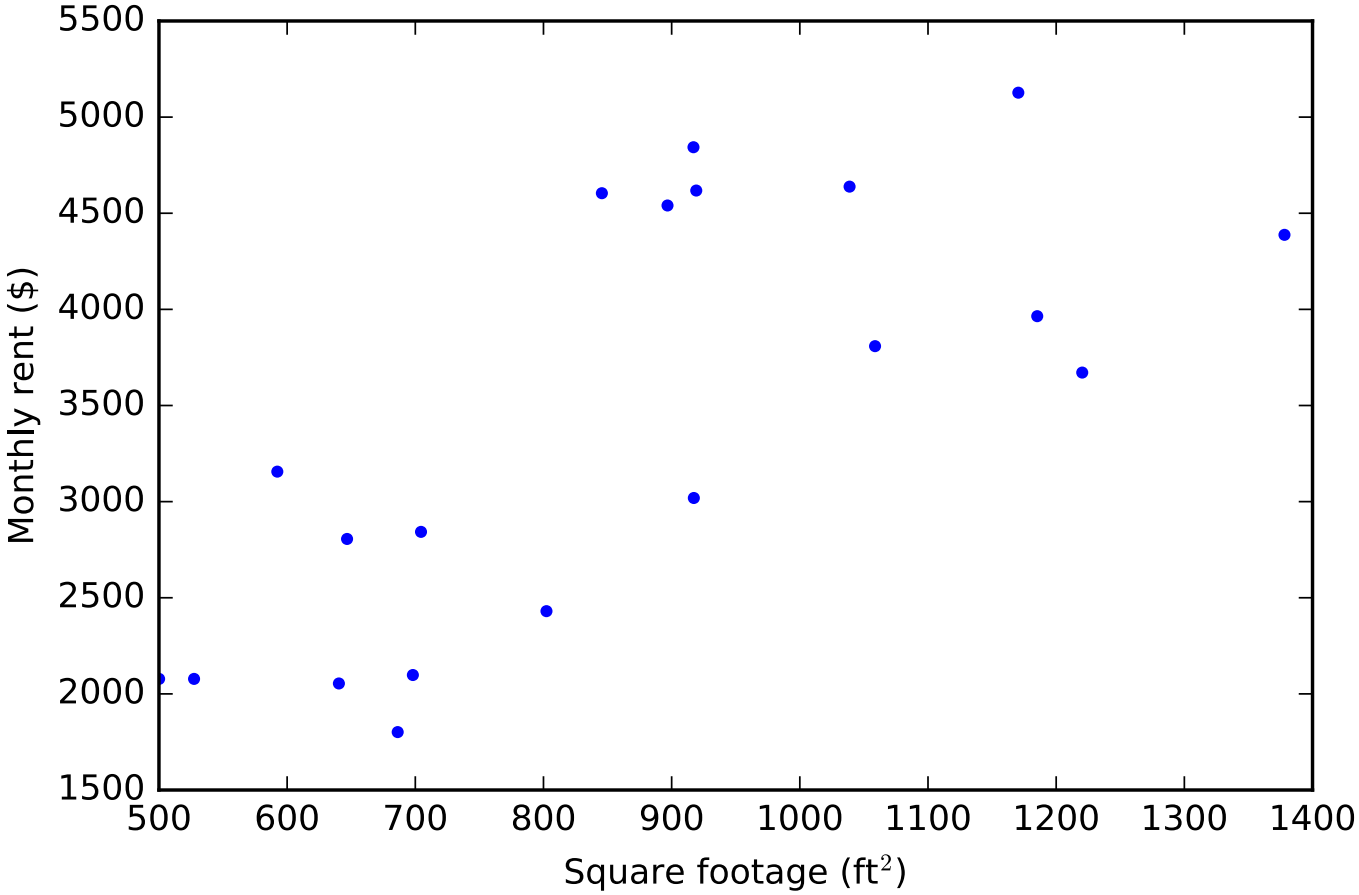
$\nabla_x y \in \mathbb{R}^{1 \times n}$

$\nabla_A y \in \mathbb{R}^{n \times m}$

In numerator layout

$$\frac{\partial y}{\partial A} = \begin{bmatrix} \frac{\partial y}{\partial a_{11}} & \frac{\partial y}{\partial a_{21}} & \cdots \\ \frac{\partial y}{\partial a_{12}} & & \\ \vdots & & \\ \frac{\partial y}{\partial a_{1n}} & & \end{bmatrix}$$

Back to our supervised learning example



Thus our strategy to find the best θ is to:

- Calculate

$$\frac{d\mathcal{L}}{d\theta}$$

- Solve for θ such that

$$\frac{\partial \mathcal{L}}{\partial \theta} = 0$$

Back to our supervised learning example

$\theta = \begin{bmatrix} a \\ b \end{bmatrix}$ $\hat{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$

Re-writing the cost function, we have:

$$\mathcal{L}(\theta) = \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \theta^T \hat{\mathbf{x}}^{(i)})^2$$

$$= \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \theta^T \hat{x}^{(i)})^T (y^{(i)} - \theta^T \hat{x}^{(i)})$$

$5^T = 5$

$(ABC)^T = C^T B^T A^T$

$$= \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \hat{x}^{(i)T}\theta)^T (y^{(i)} - \hat{x}^{(i)T}\theta)$$

"vectorization"

$$= \frac{1}{2}\left(\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} - \begin{bmatrix} \hat{x}^{(1)T} \\ \hat{x}^{(2)T} \\ \vdots \\ \hat{x}^{(N)T} \end{bmatrix}\theta\right)^T \left(\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} - \begin{bmatrix} \hat{x}^{(1)T} \\ \hat{x}^{(2)T} \\ \vdots \\ \hat{x}^{(N)T} \end{bmatrix}\theta\right)$$

$Y \in \mathbb{R}^N$ $X \in \mathbb{R}^{N\times 2}$ $\theta \in \mathbb{R}^2$

$$= \frac{1}{2}(Y - X\theta)^T(Y - X\theta) = \frac{1}{2}\|Y - X\theta\|_2^2$$

$$= \frac{1}{2}(Y^TY - Y^TX\theta - \theta^TX^TY + \theta^TX^TX\theta)$$

$A = (BC)$

$A^T = C^T B^T$

Back to our supervised learning example

$\frac{\partial \theta^T A \theta}{\partial \theta} = (A + A^T)\theta$

$w = z^T \theta \quad \frac{\partial w}{\partial \theta} = z$

Let's now take derivatives:

$$\mathcal{L}(\theta) = \frac{1}{2}\left(Y^T Y - 2\overbrace{Y^T}^{z^T}\mathbf{X}\theta + \theta^T\overbrace{\mathbf{X}^T\mathbf{X}}^{A}\theta\right)$$

$(1\times N)(N\times 2)$ $\rightarrow 1\times 2$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{1}{2}\left[0 - 2X^T Y + (X^T X + X^T X)\theta\right]$$

$$= -X^T Y + X^T X \theta$$

$$[=]\ 0$$

$$X^T Y = X^T X \theta$$

$$\Rightarrow \quad \theta = (X^T X)^{-1} X^T Y$$

Back to our supervised learning example

Back to our supervised learning example

This solution is called least-squares, and it appears in a variety of linear applications.

$$\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T Y$$

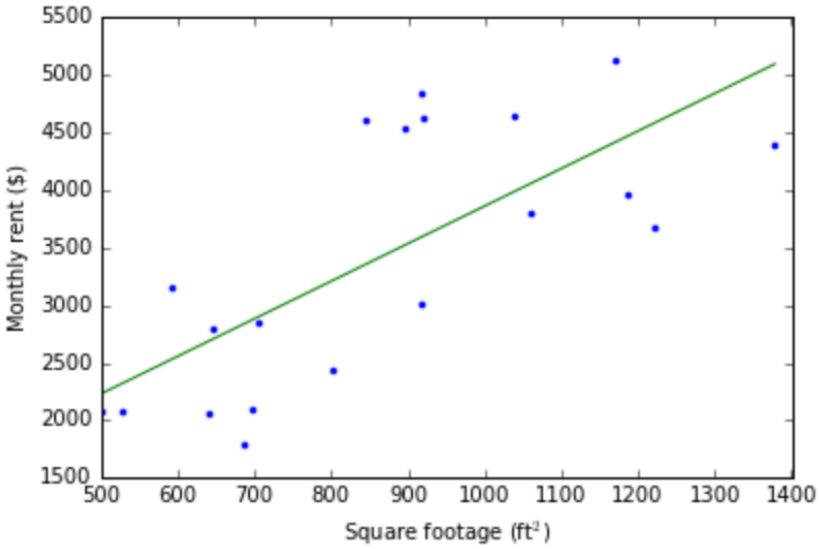
Back to our supervised learning example

y (N,) array

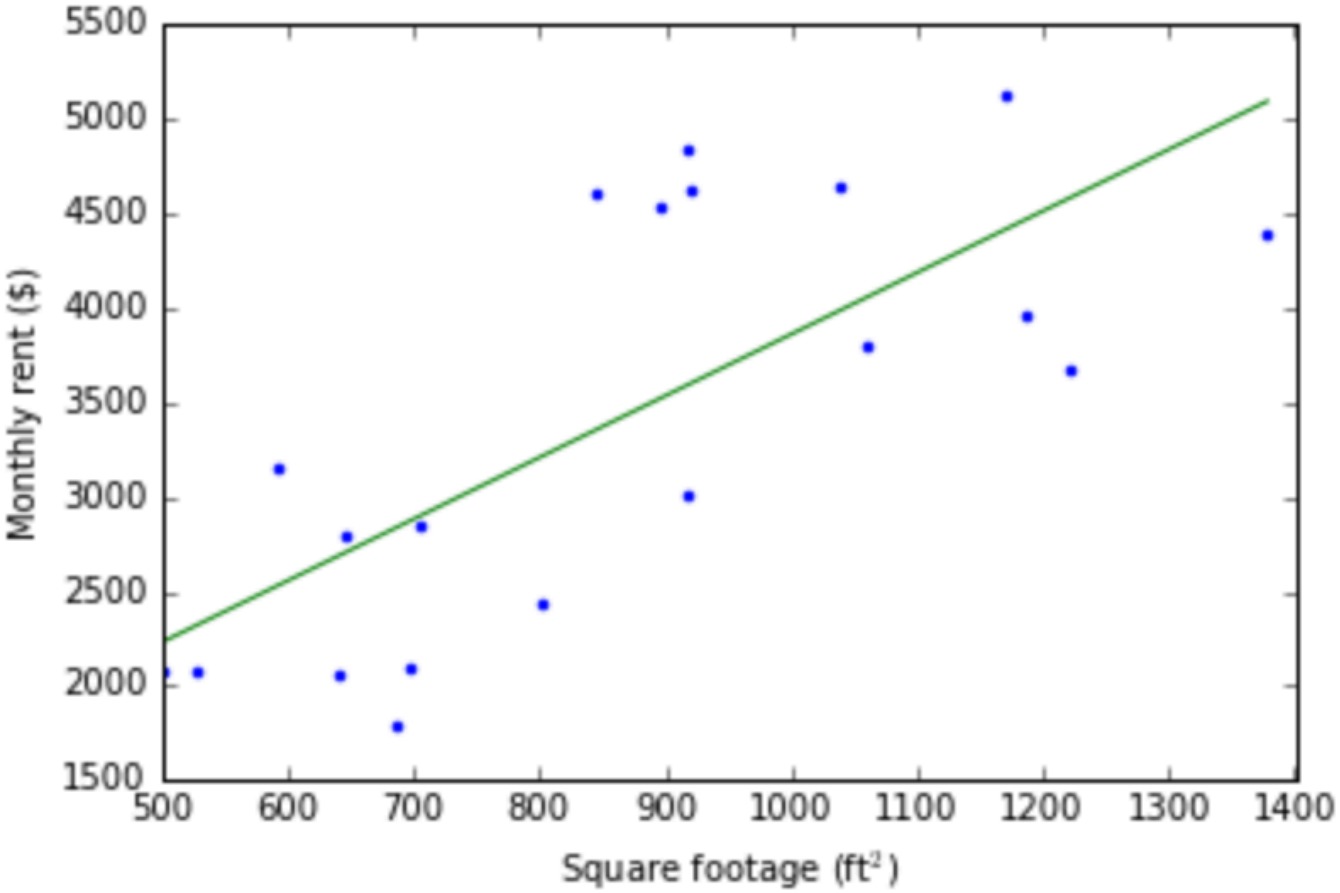
x (N,) array

```
# Given paired data, with x being square footages and y being monthly rents
# Fit a linear model
xhat = np.vstack((x, np.ones_like(x)))
theta_lin = np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y))

# Plot the data
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('Square footage (ft$^2$)')
ax.set_ylabel('Monthly rent ($)')
f.savefig('ml-basics_rent.pdf')

# Plot the regression line
xs = np.linspace(min(x), max(x),50)
xs = np.vstack((xs, np.ones_like(xs)))
plt.plot(xs[0,:], theta_lin.dot(xs))

print theta_lin
```

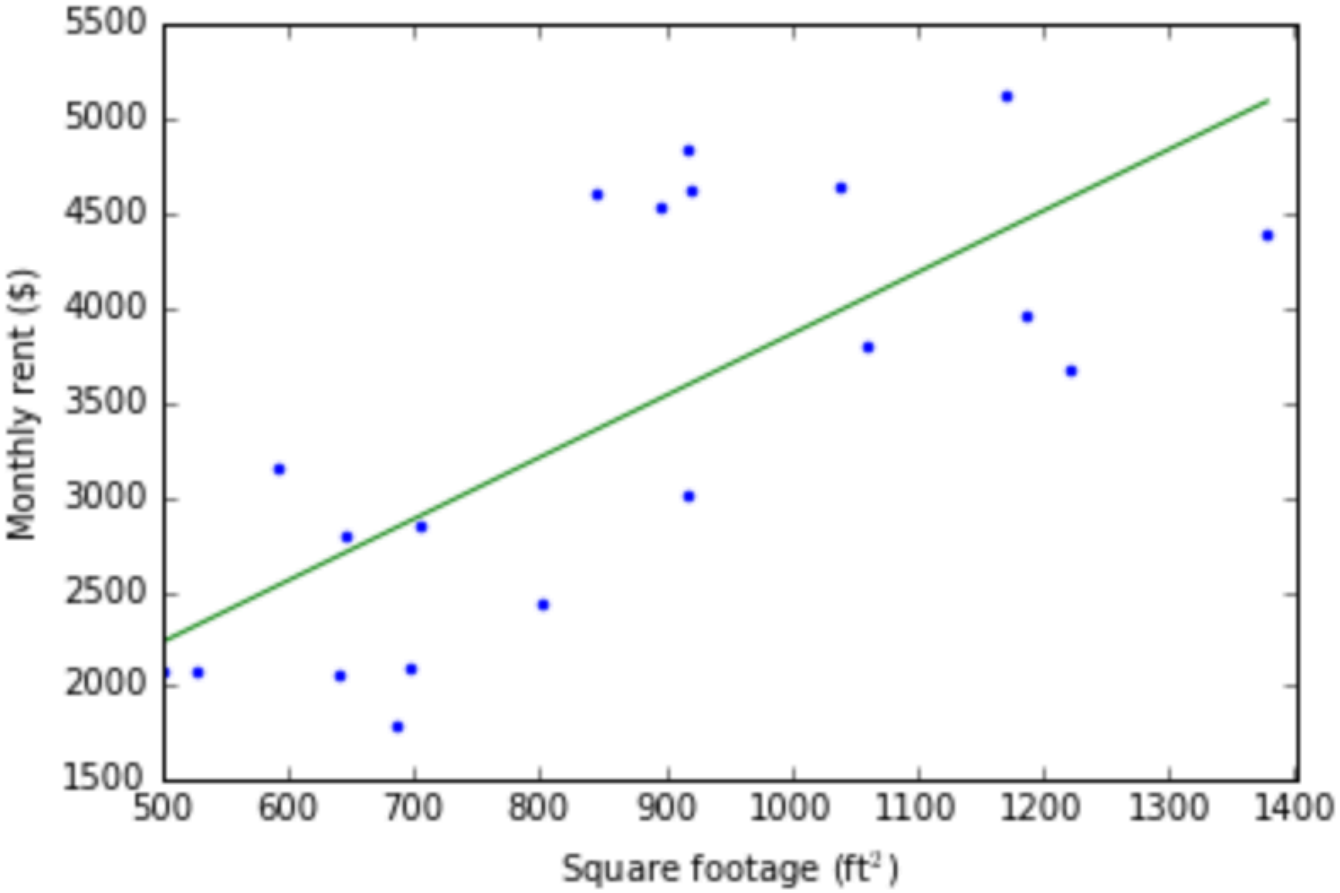
```
[   3.25900793  601.85544895]
```



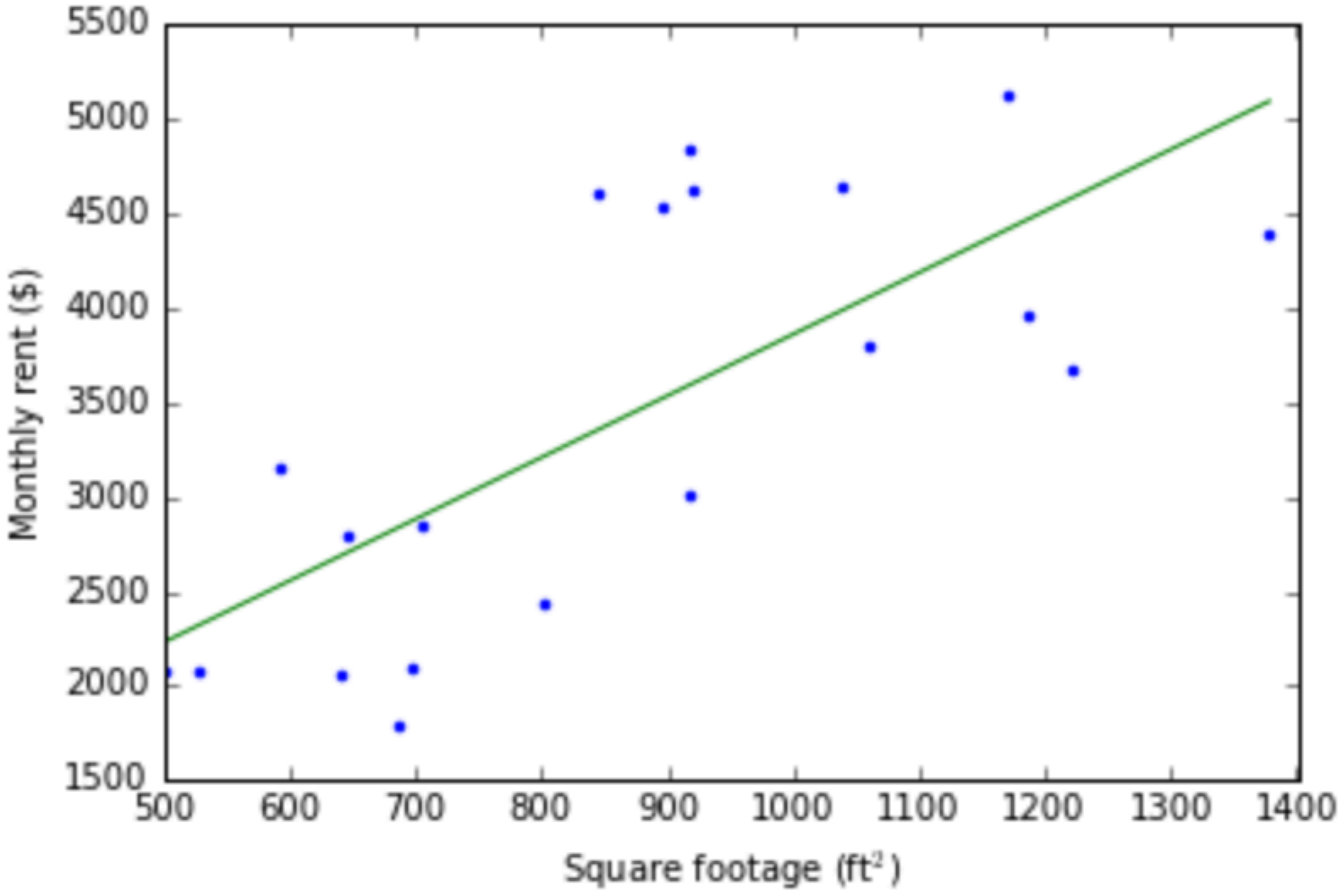
Can’t we do better?



Can’t we do better?



Question:



Question: How does our current least-squares formula allow for learning nonlinear polynomial fits, e.g.,

$$y = b + a_1 x_1 + a_2 x^2 + \cdots + a_n x^n$$

$$\hat{x} = \begin{bmatrix} x^3 \\ x^2 \\ x \\ 1 \end{bmatrix}$$

More generally…

With our problem setup, it's straightforward to generalize to higher degree polynomial models, e.g.,

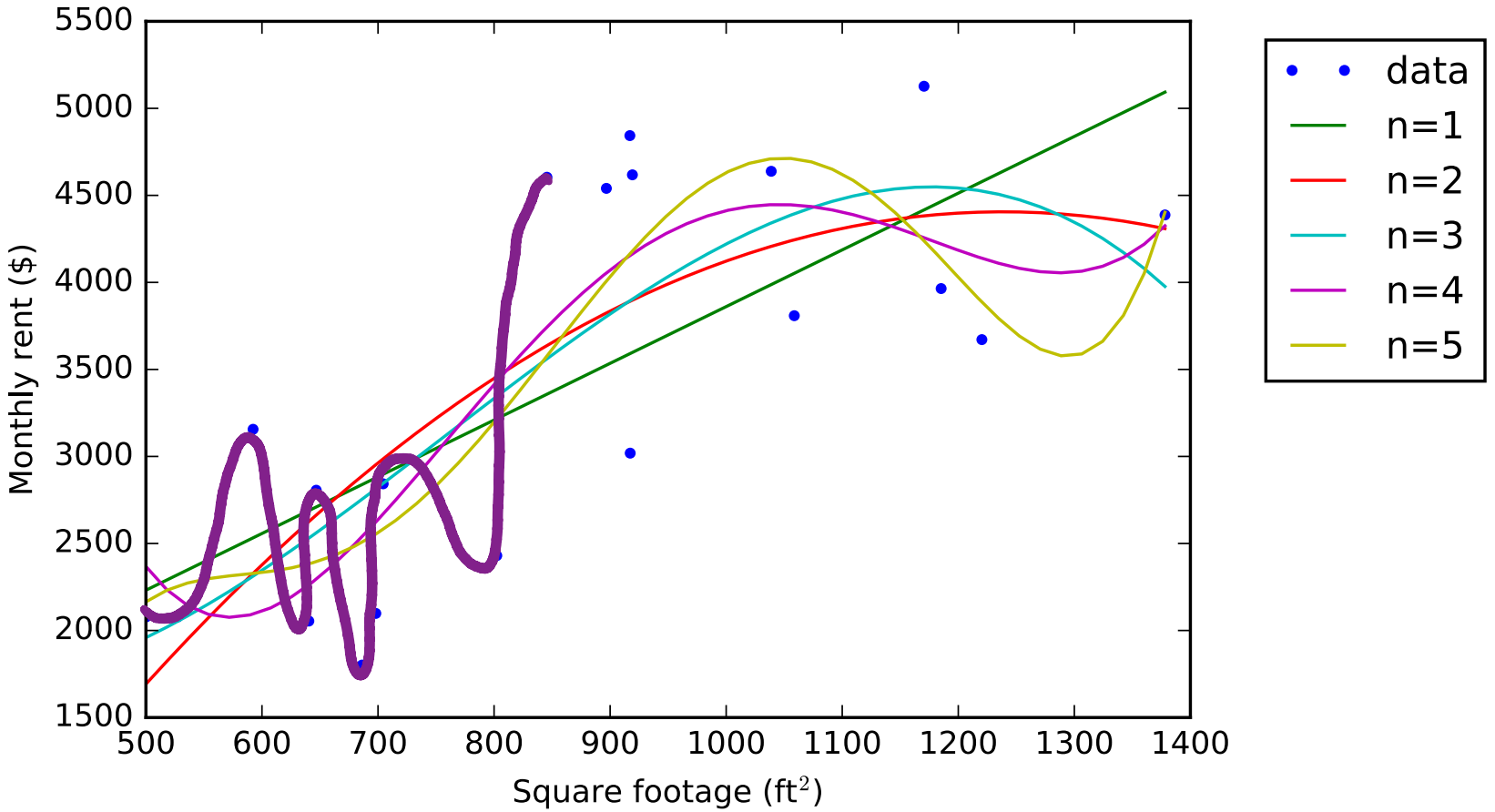
$$y = b + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

$$\hat{x} = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^n \end{bmatrix} \qquad \theta = \begin{bmatrix} b \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

$$= \theta^T \hat{x}$$

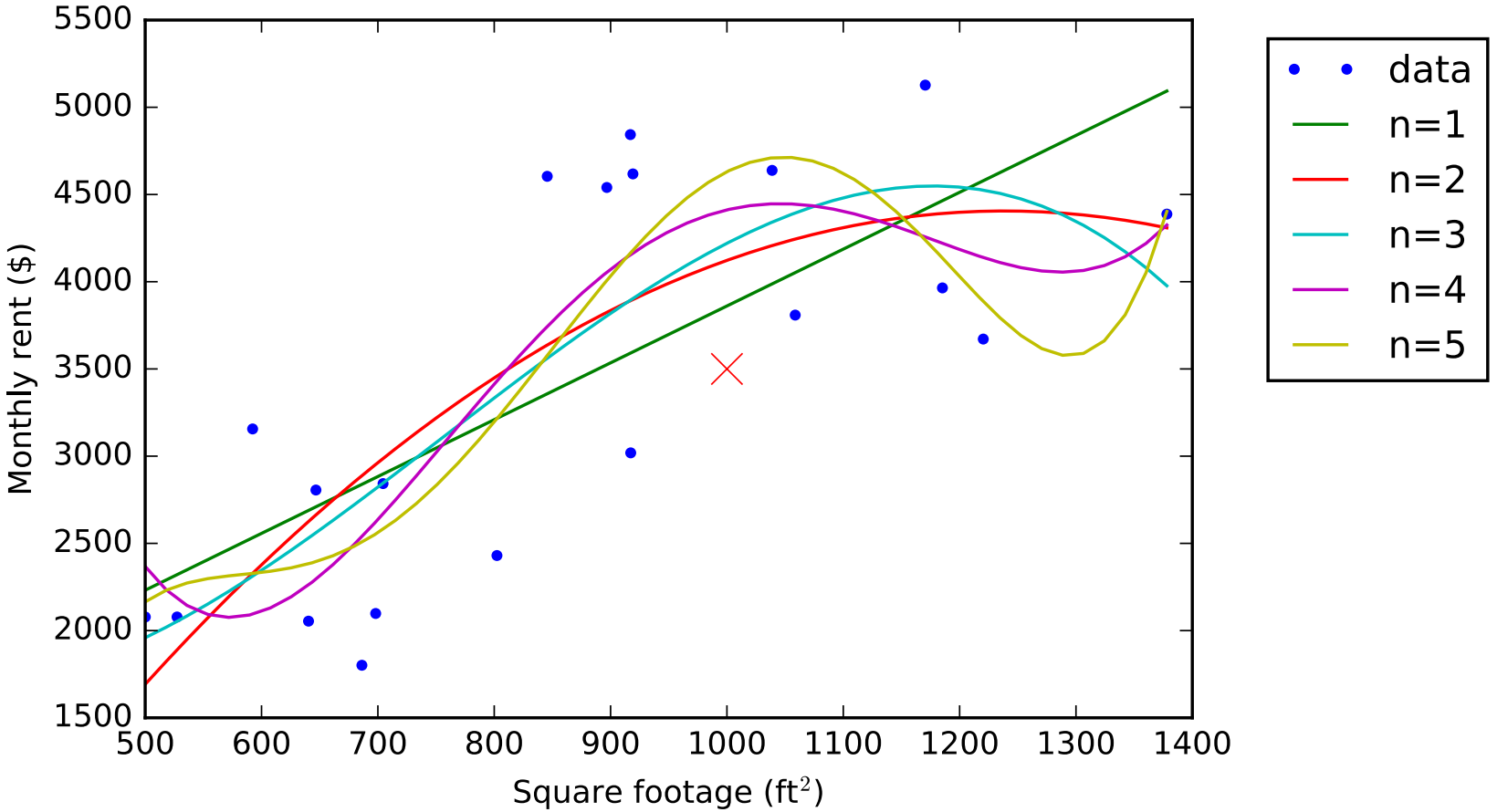
More generally…

$$y = b + a_1 x + a_2 x^2 + a_3 x^3$$



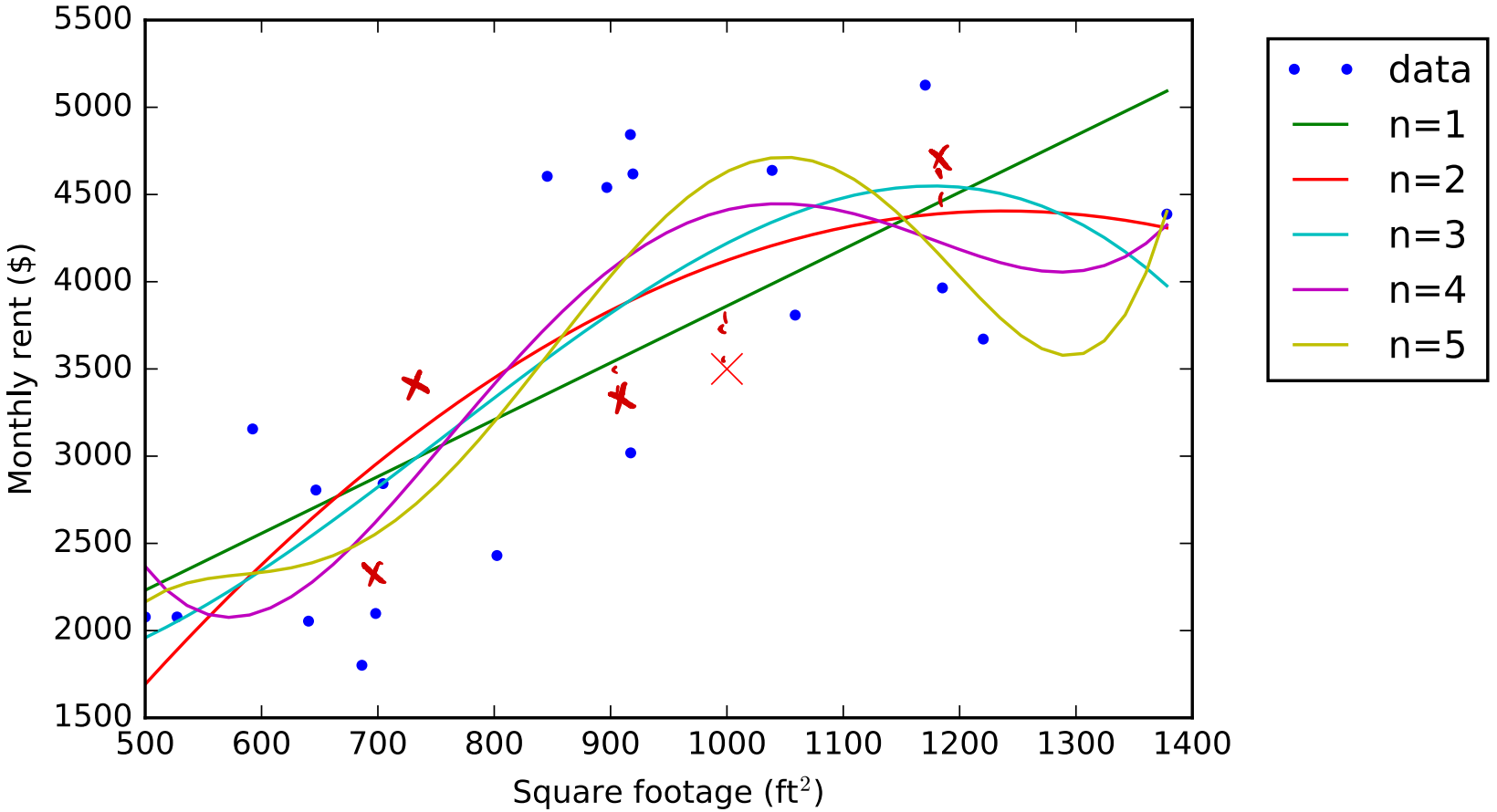
Now, a higher degree polynomial will *always* fit the **provided** data as well as a lower order polynomial. Why?

Now, let's say a new house pops up for rent…



In this scenario, the linear model best predicts the price of the new house ('x')

Model generalization



The fundamental problem here is that the more complex models may **not generalize as well** if the data come from a different model.

Training and testing data

Overfitting

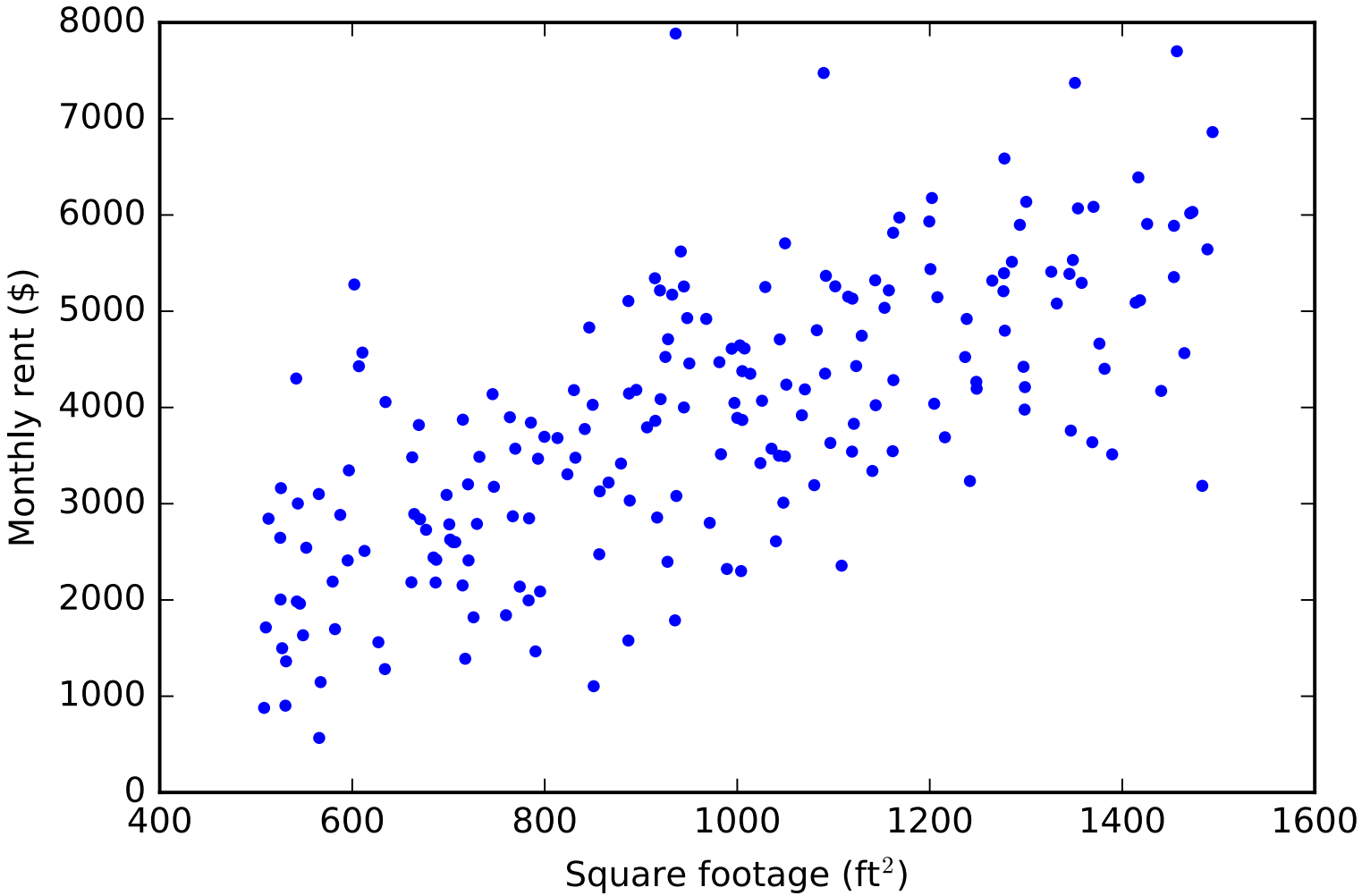
This idea of generalization can be made more formal by introducing the concepts of a *training set* and *testing set*.

- **Training data** is data that is used to learn the parameters of your model.
- **Testing data** is data that is excluded in training and used to score your model.

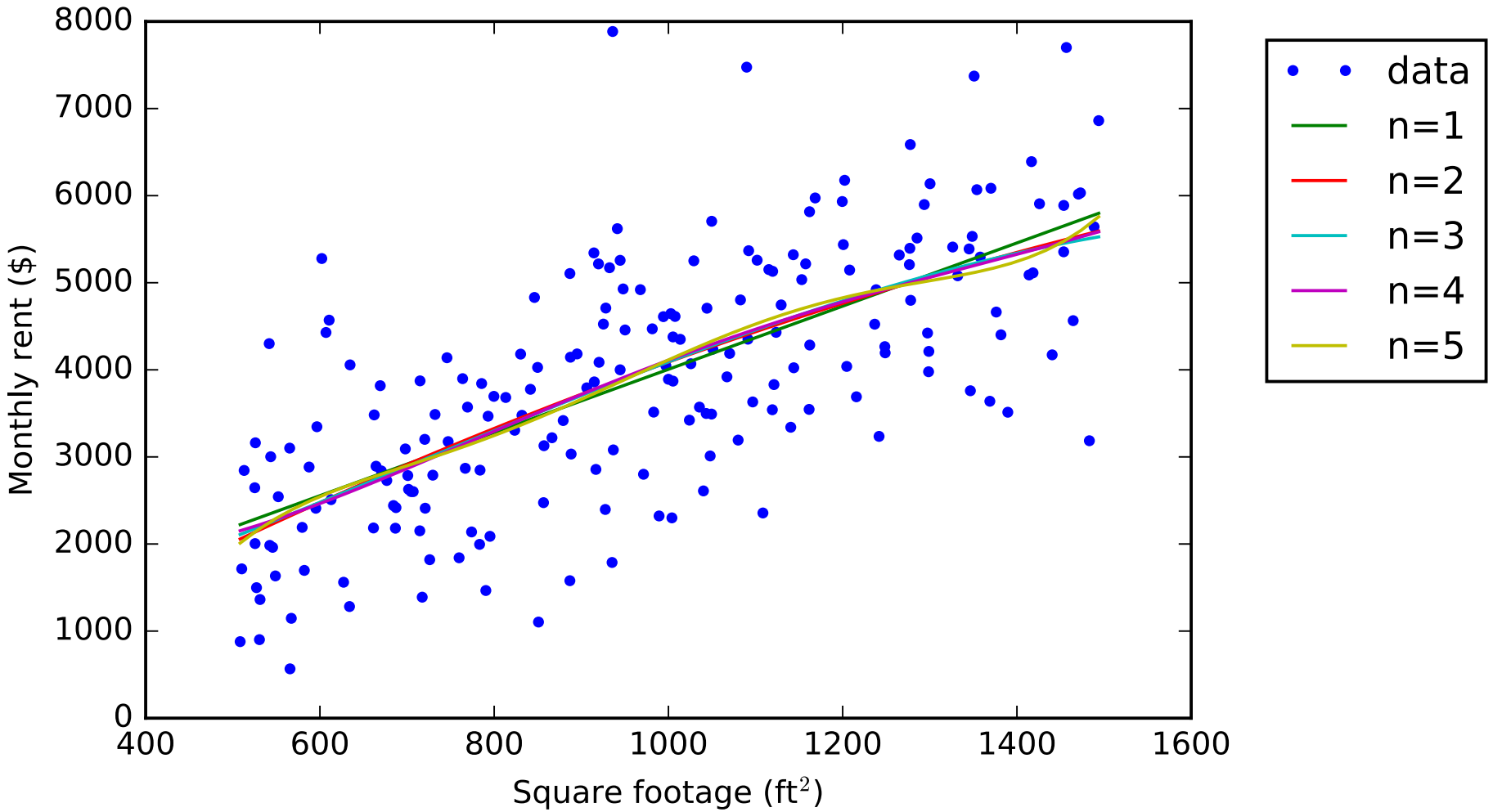
There is also a notion of validation data, which we will get to later.

A model which has very low training error but high testing error is called *overfit*.

More data helps to avoid overfitting



More data helps to avoid overfitting



More data helps to avoid overfitting

This suggests that when *a lot* of data is available, it may be appropriate to use more complex models. (Another technique we will discuss later, regularization, also helps with overfitting.) This is a shadow of things to come (i.e., neural networks which have large complexity).

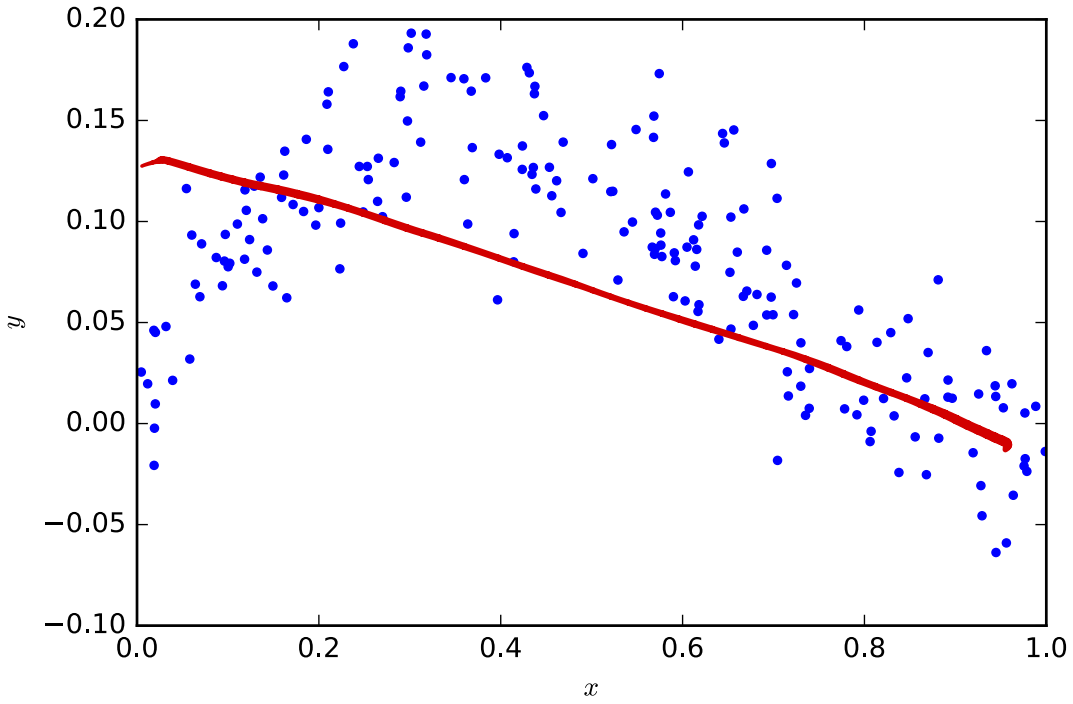
Underfitting

At the same time, we can not make the model *overly simple* as then we will underfit the data. This corresponds to a model that has both high training and high testing error, and the fit generally will not improve with more training data because the model is not expressive enough.

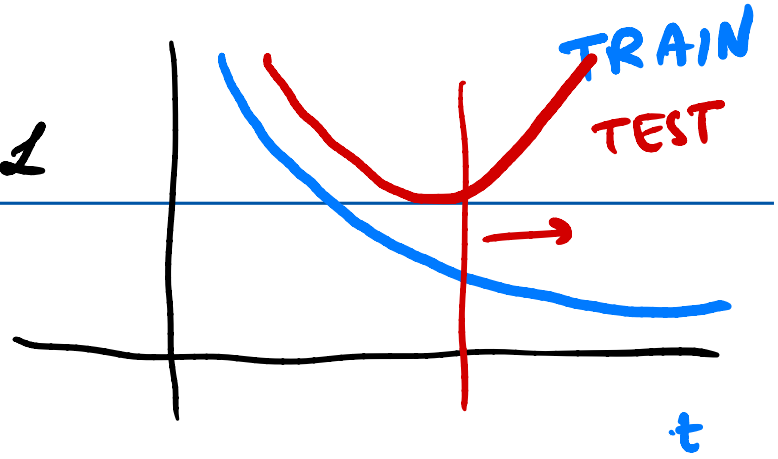
Underfitting

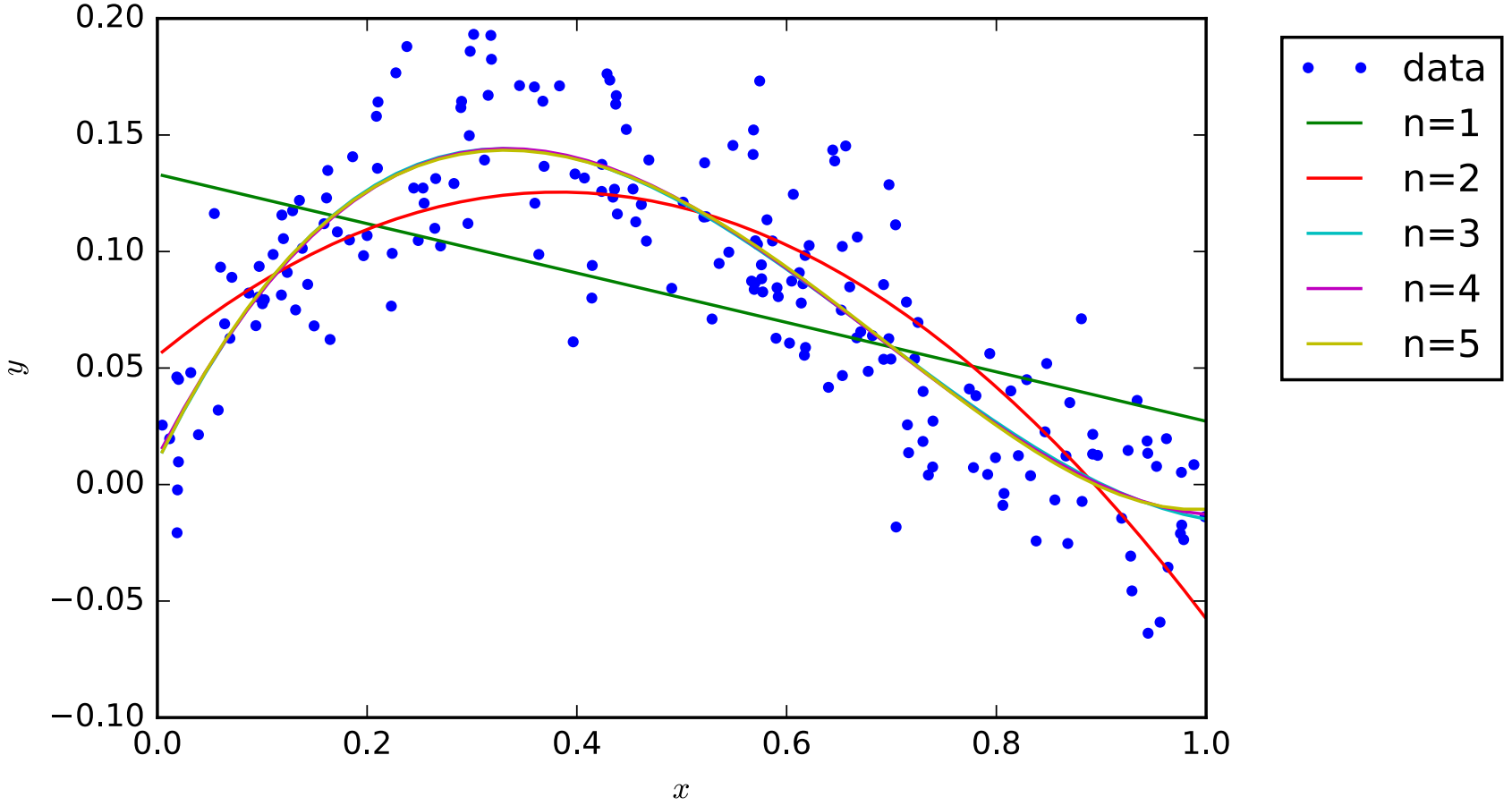
$y = ax + b$

```
np.random.seed(0)  # Sets the random seed.
num_train = 200      # Number of training data points

# Generate the training data
x = np.random.uniform(low=0, high=1, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_train,))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
f.savefig('ml-basics_underfitting-data.pdf')
```



Underfitting





Evaluating generalization error

Training, validation, and testing data

The standard for training, evaluating, and choosing models is to use different datasets for each step.

- **Training data** is data that is used to learn the parameters of your model.
- **Validation data** is data that is used to optimize the hyperparameters of your model. This avoids the potential of overfitting to nuances in the testing dataset.
- **Testing data** is data that is used to score your model.

Note, in many cases, testing and validation datasets are used interchangeably as datasets that are used to evaluate the model. In this scenario, hyperparameters would also be optimized using training data.

Evaluating generalization error

k-fold cross validation

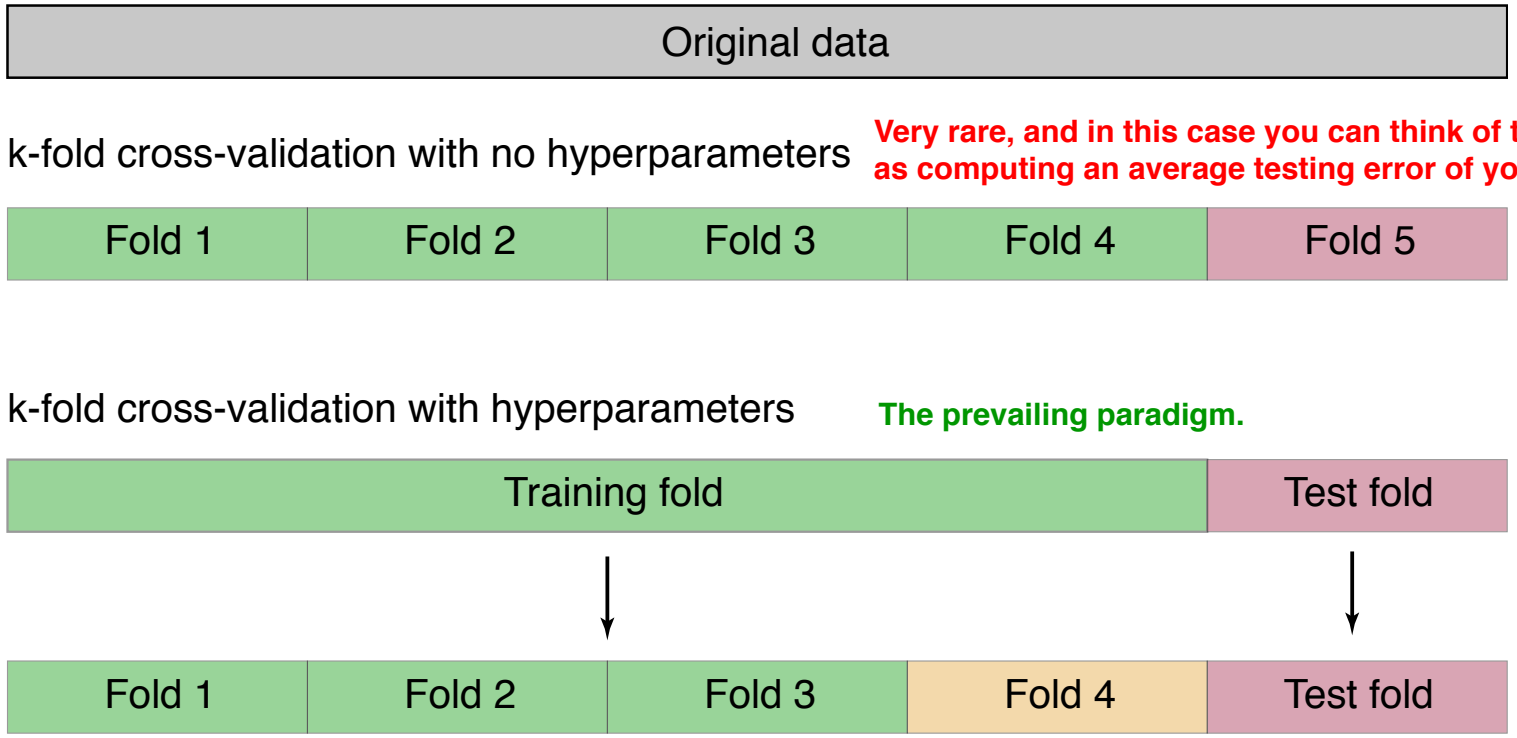
This assumes you already have a *SEPARATE* testing set.
The TRAINING set is split into training and validation data.

4-fold CV

In a common scenario, you will be given a training dataset and a testing dataset. To train a model using this dataset, one common approach is k-fold cross validation. Then the procedure looks as follows:

- Let the training dataset contain N examples. N=800
- Then, split the data into k equal sets, each of N/k examples. Each of these sets is called a "fold." 4 folds, each with 200 examples
- $k-1$ of the folds are datasets that are used to train the model parameters.
- The remaining fold is a ~~testing~~ validation dataset used to evaluate the model.
- You may repeatedly train the model by choosing which folds comprise the training folds and the testing fold.

Evaluating generalization error



Evaluating generalization error

Original data

N = 1000

k-fold cross-validation with hyperparameters

The prevailing paradigm.

200 ex

Training fold 800 ex. | Test fold

4-fold CV

Validation

Fold 1	Fold 2	Fold 3	Fold 4	Test fold
200	200	200	200	

learn θ

generalize to unseen data