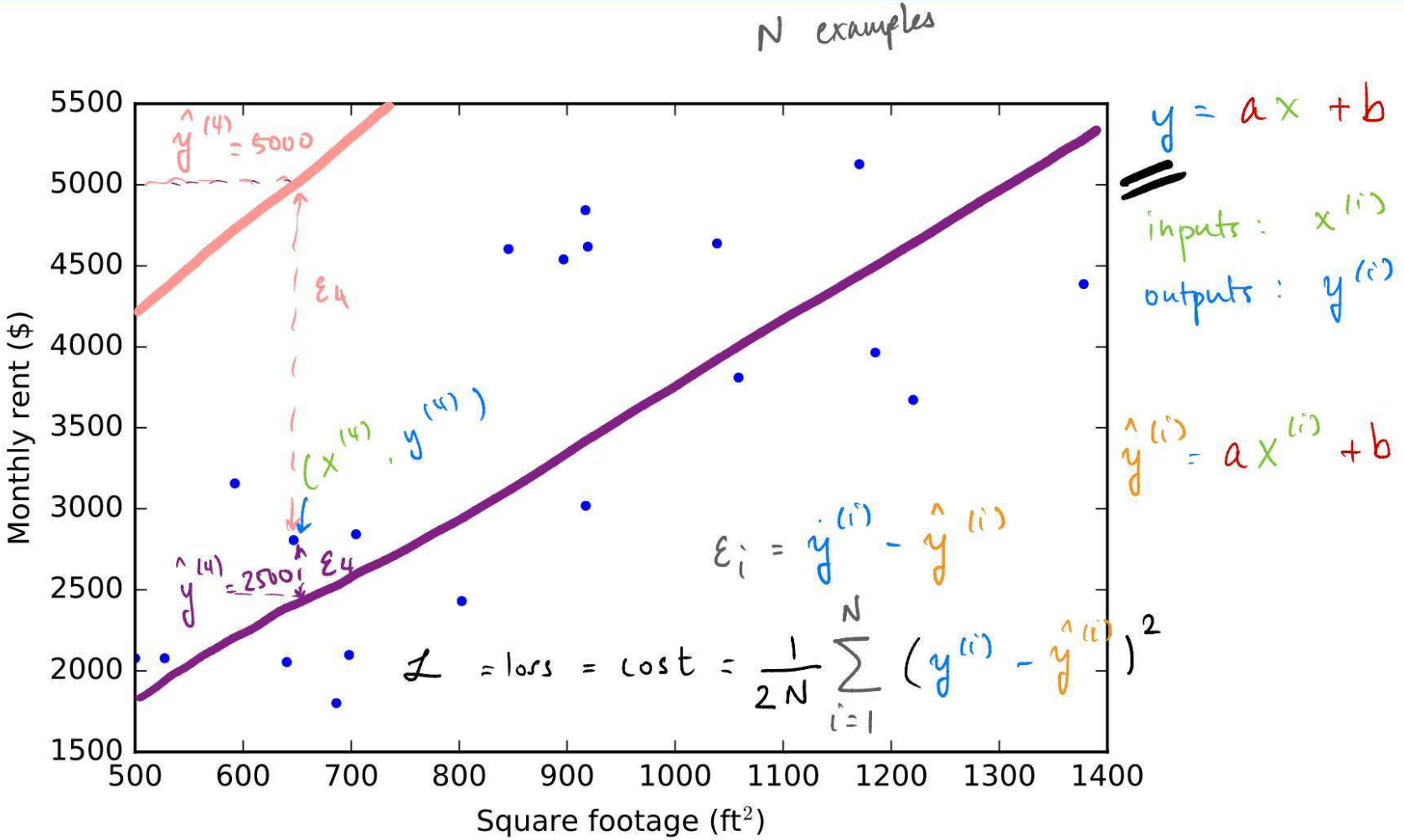
Lecture 3: ML Refresher and Supervised Classification

Announcements:

- HW #1 is due Monday Jan 20, uploaded to Gradescope. To submit your Jupyter Notebook, print the notebook to a pdf with your solutions and plots filled in. The graders will grade your work from these notebooks. In the future, you will also have to print out code from any .py files that were modified.

- Zoom logistics.

An example of supervised learning



How should we model this data?

- Inputs, **x**? Outputs, **y**? Data
- What model should we use? f
- How do we assess how good our model is? $\mathcal{L}$

An example of supervised learning

$\hat{y} = ax^2 + bx + c$

$\theta = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \hat{x} = \begin{bmatrix} x^2 \\ x \\ 1 \end{bmatrix}$

A simple linear example:

- Model:

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

y: data given to us

parameters: I get to choose a, b to make my loss as small as possible (this makes the model as good as possible.)

- Cost function:

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

$\mathcal{L}(a, b)$

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

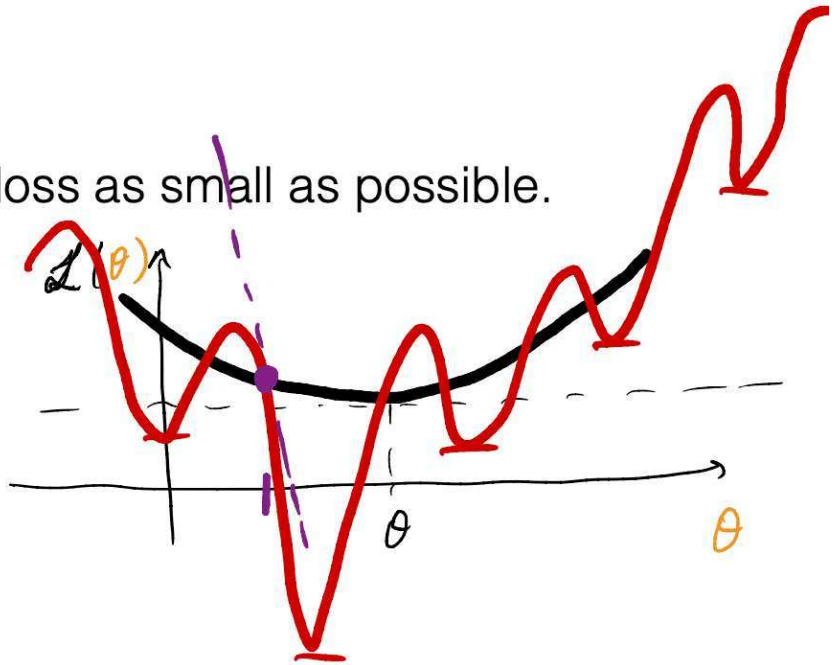
How do we learn the parameters of this model?

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.

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



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

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Example: derivative with respect to a vector

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

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

$y = \theta^T x$ $\qquad \nabla_x y$ $\qquad y \in \mathbb{R}$

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

$$y = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

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

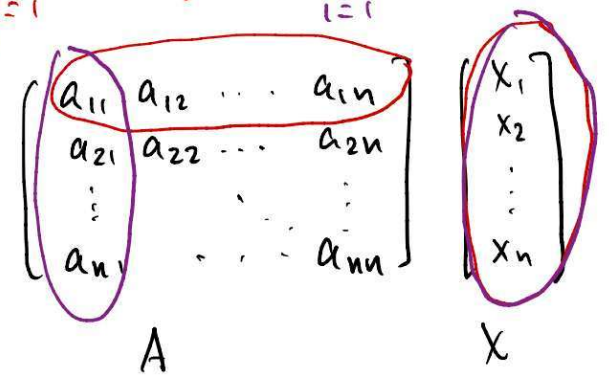
$$\nabla_x y = \theta$$

$$\nabla_\theta y = x$$

$y = x^T A x$

Aside: vector and matrix derivatives

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



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

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

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

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

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

if A is symmetric $\Rightarrow = 2Ax$

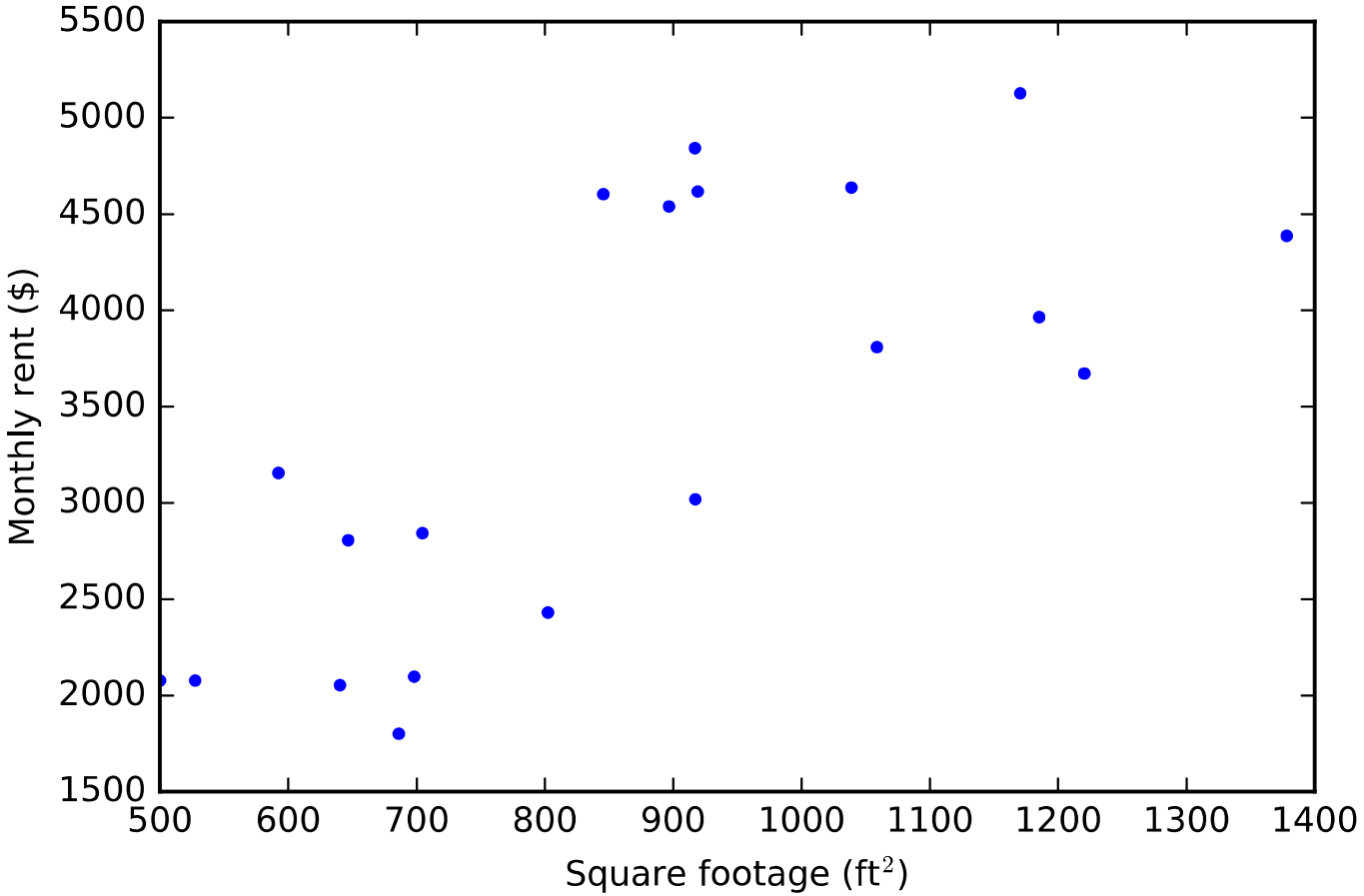
$(n \times n)(n \times 1)$

when $n = 1$

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

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

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

Re-writing the cost function, we have:

$5^T = 5$

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

$(\theta^T \hat{x}^{(i)})^T = \hat{x}^{(i)T} \theta$

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

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

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

"vectorization" HW #2

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

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

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

$$= \frac{1}{2N} \left(Y^T Y - \underbrace{Y^T X \theta}_{(1 \times N)(N \times 2)(2 \times 1)} - \theta^T X^T Y + \theta^T X^T X \theta \right)$$

$\nabla_\theta \theta^T A \theta = (A + A^T)\theta$

Back to our supervised learning example

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

Let's now take derivatives:

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

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

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

$$= \frac{1}{2N}\left(-2X^T Y + 2X^T X \theta\right)$$

$[=]\ 0$

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

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

"Least-squares."

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

$X^T X \in \mathbb{R}^{2 \times 2}$

Back to our supervised learning example

Back to our supervised learning example

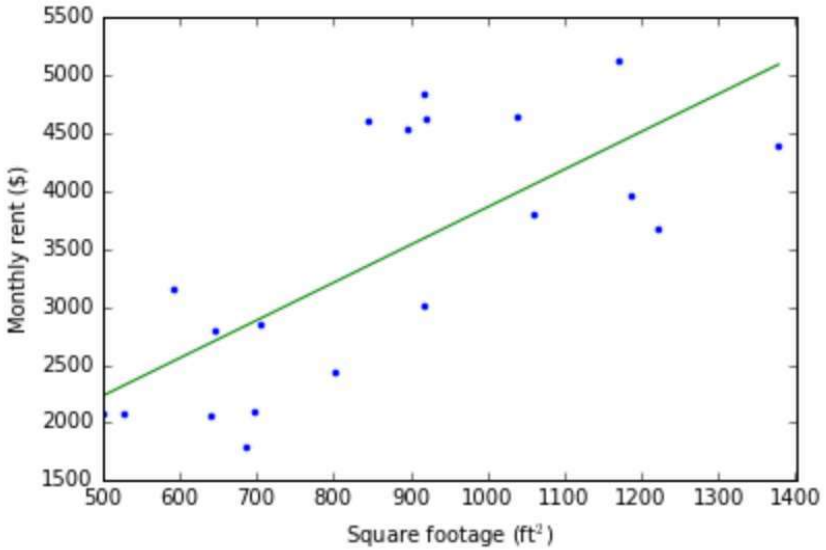
This solution is called least-squares, and it appears in a variety of linear applications.

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

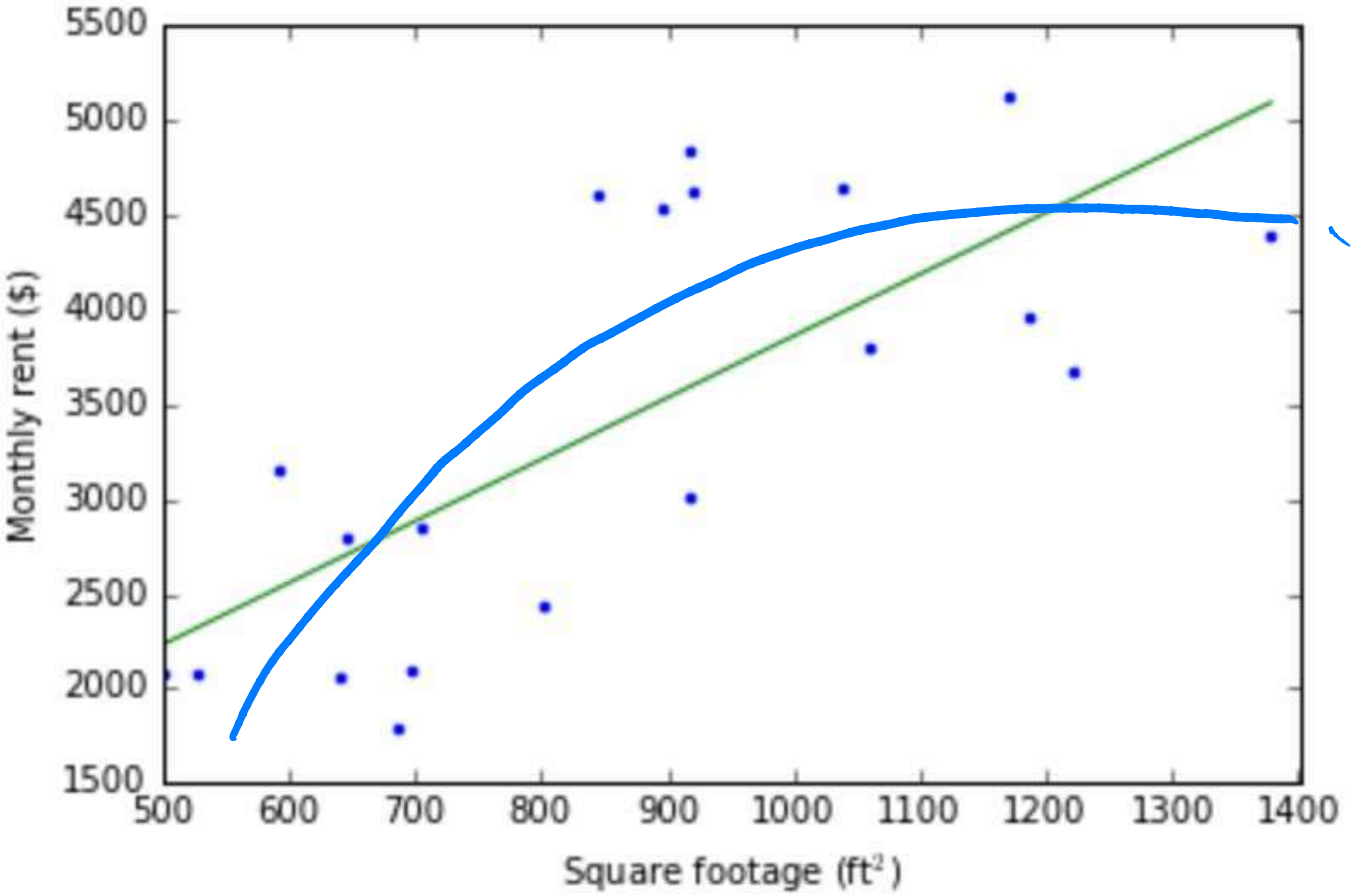
Back to our supervised learning example

```
# 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
```

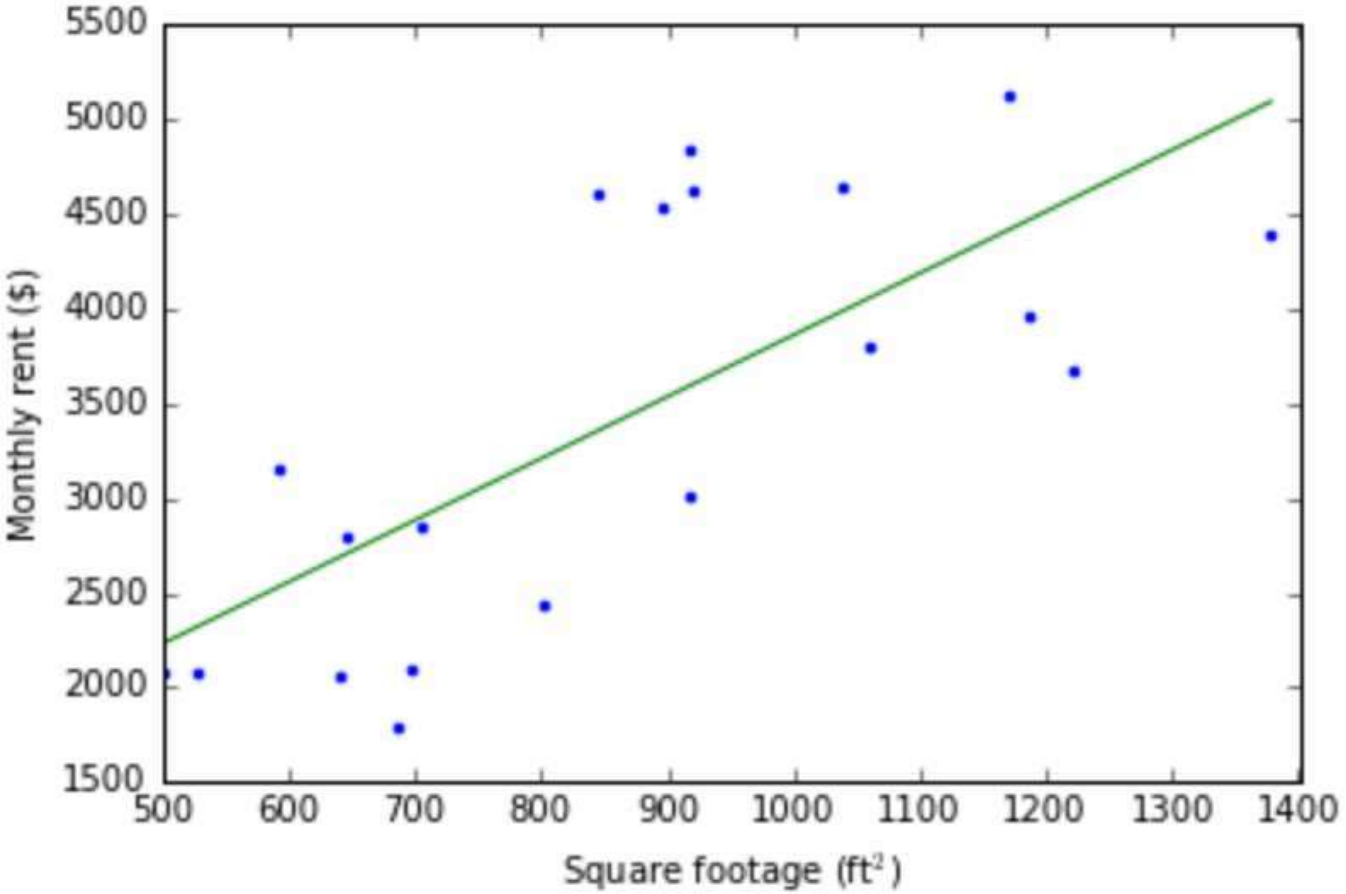
```
[   3.25900793  601.85544895]
```



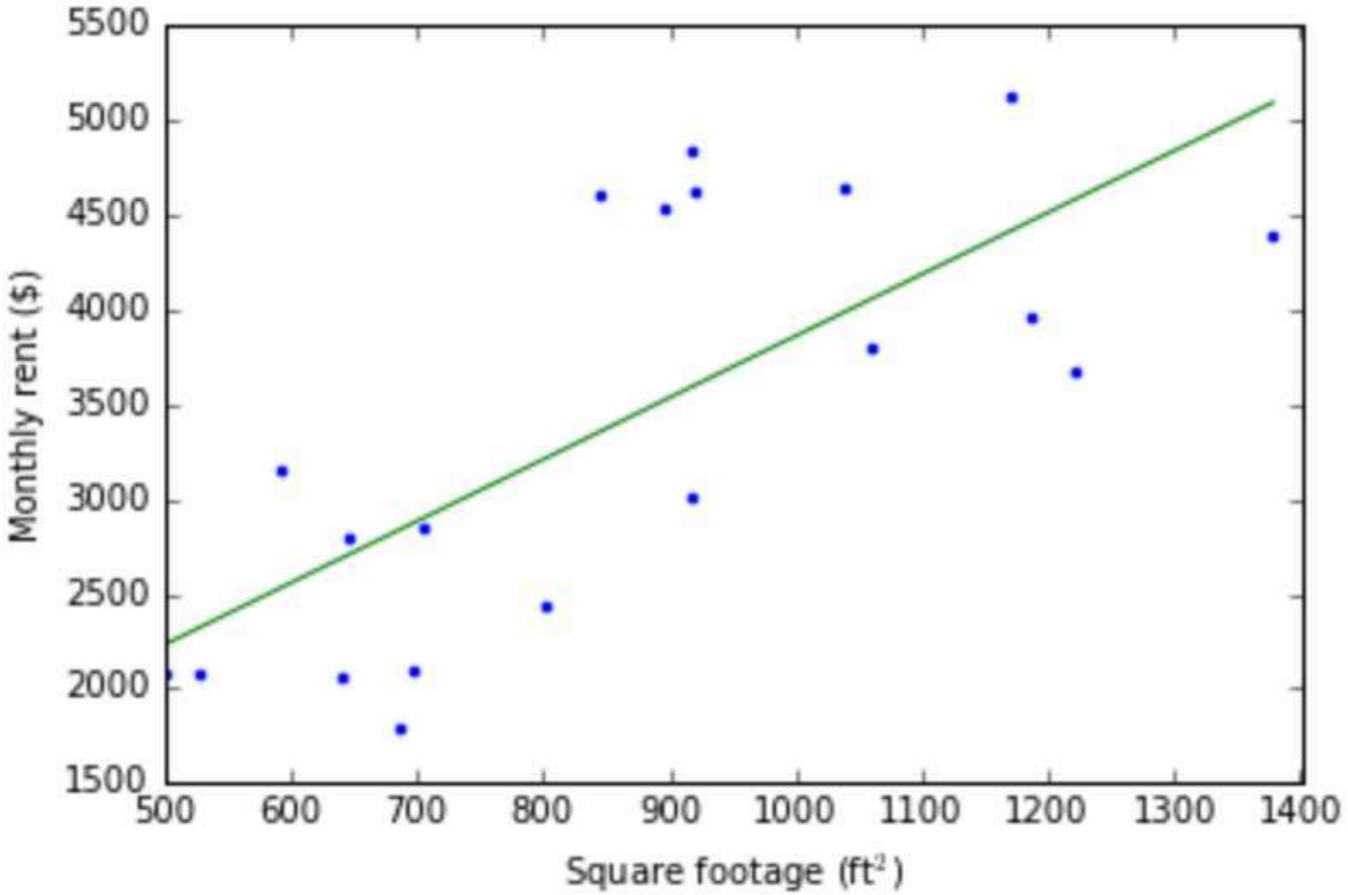
Can’t we do better?



Can’t we do better?



Question:



$n = 3$

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

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

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

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

$\theta = \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ b \end{bmatrix}$

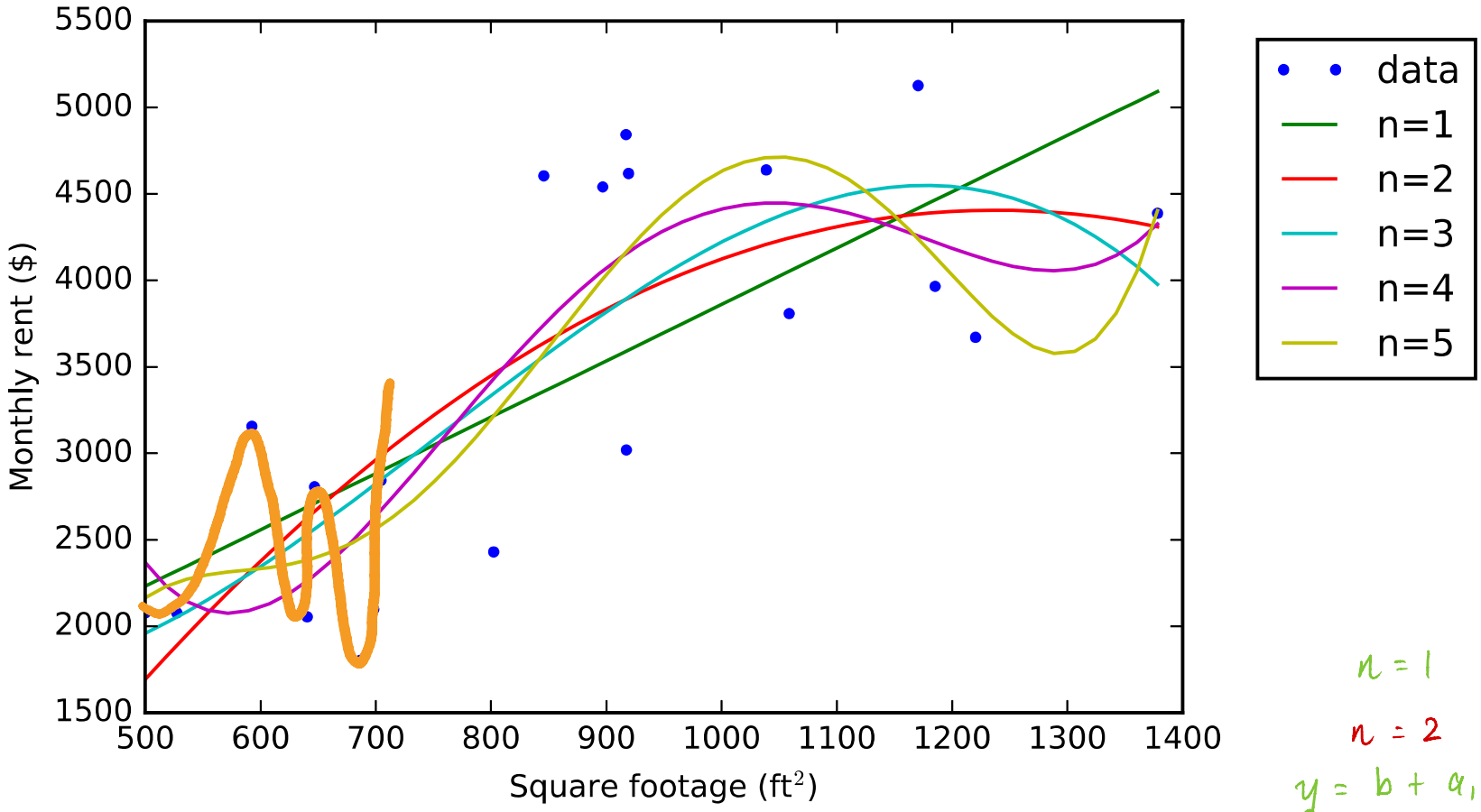
More generally…

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

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

More generally…

$y = b + a_1 x + a_2 x^2 + \cdots + a_{1000} \cdot x^{1000}$



$n = 1$

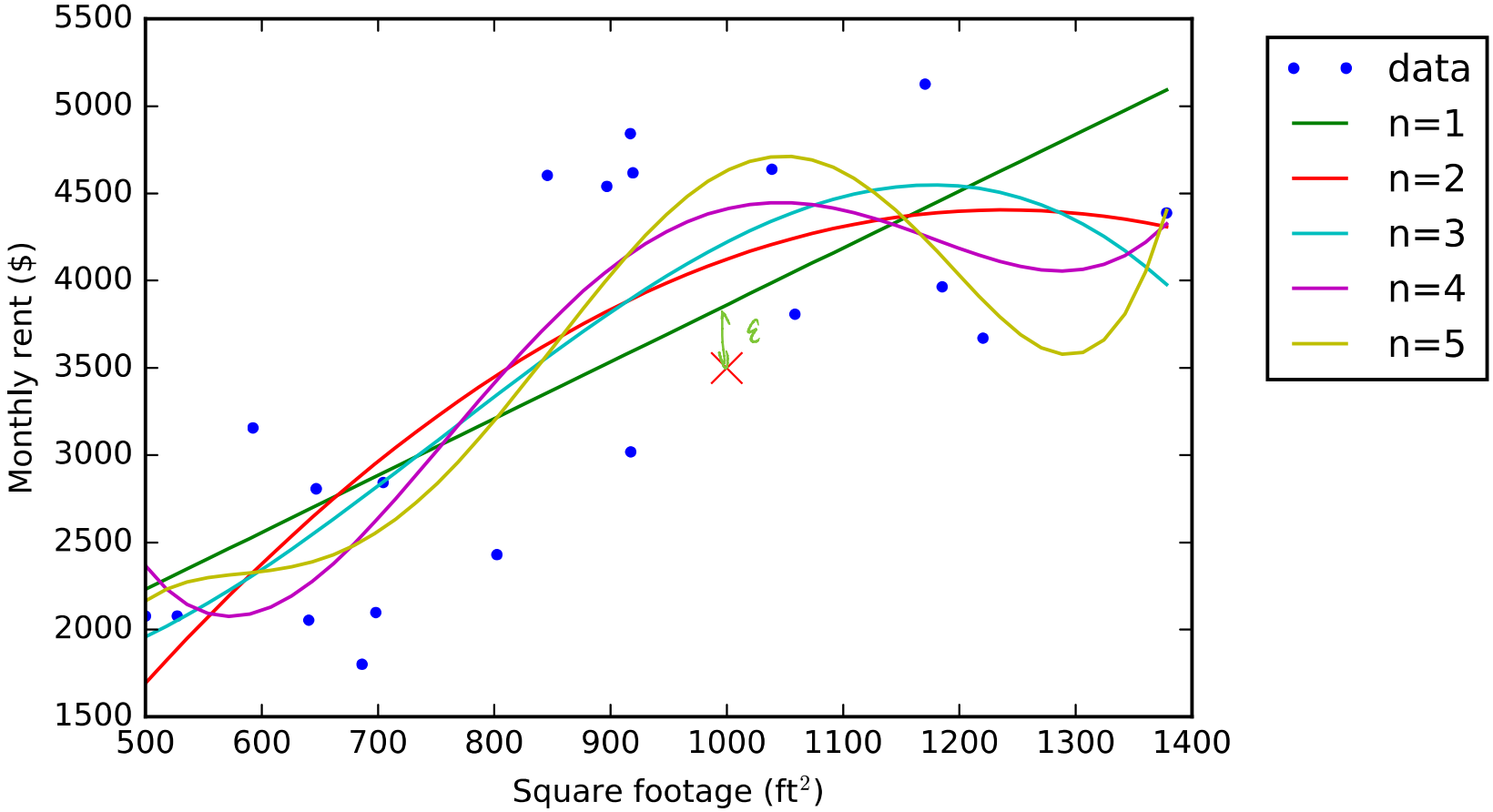
$n = 2$

$y = b + a_1 x$

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

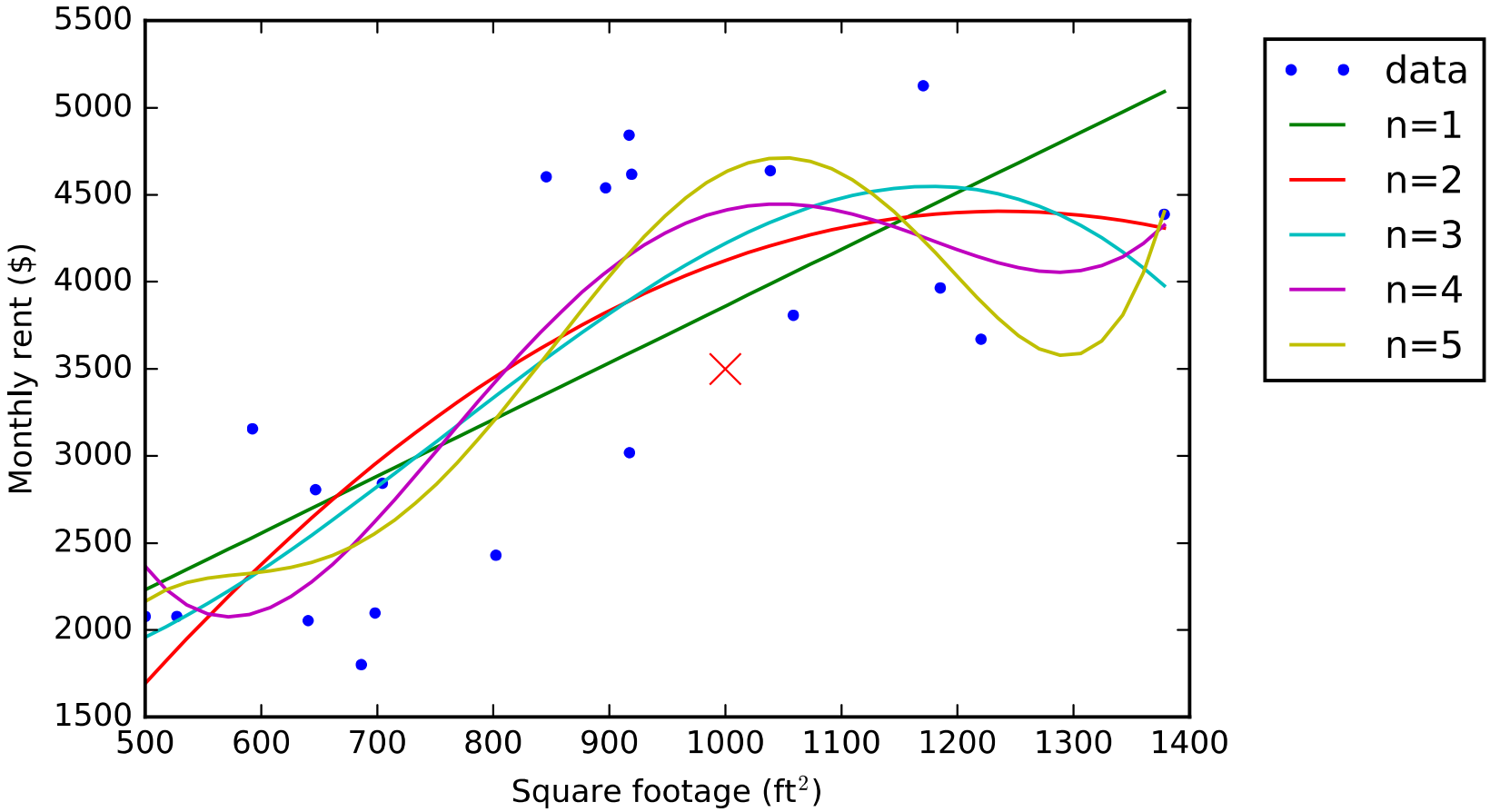
Now, a higher degree polynomial will *always* fit the **provided** data as well as a lower order polynomial. Why?

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



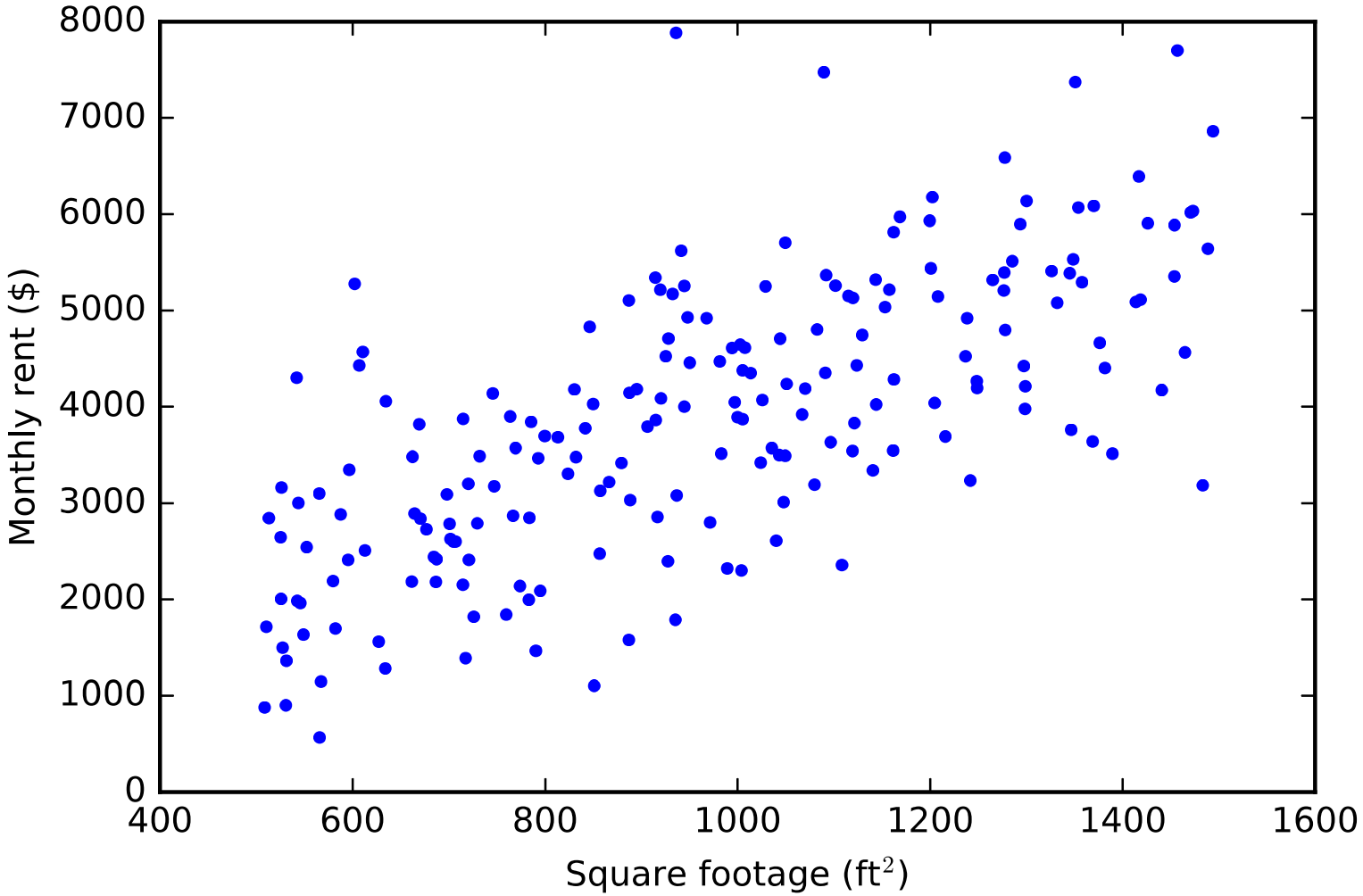
In this scenario, the linear model best predicts the price of the new house ('x')

Model generalization

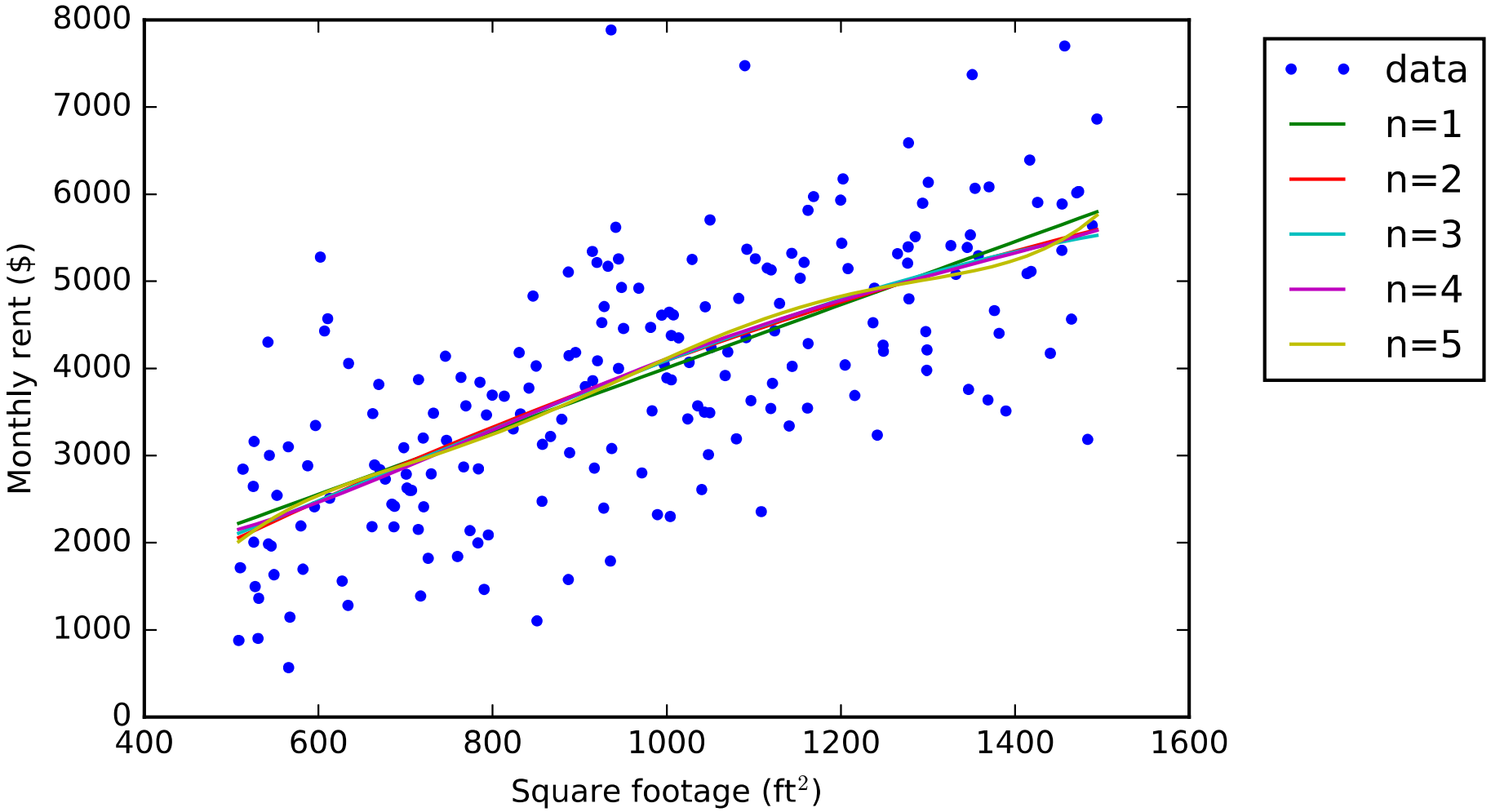


The fundamental problem here is that the more complex models may **not generalize as well** if the data come from a different model.

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).

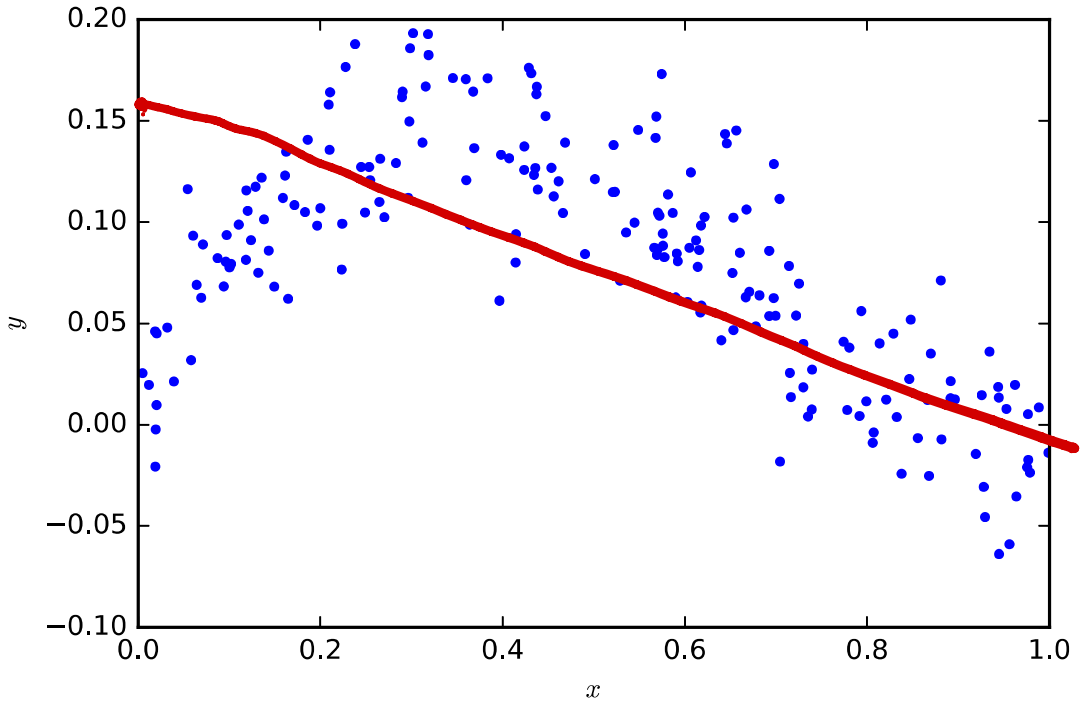
Underfitting

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

```
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')
```

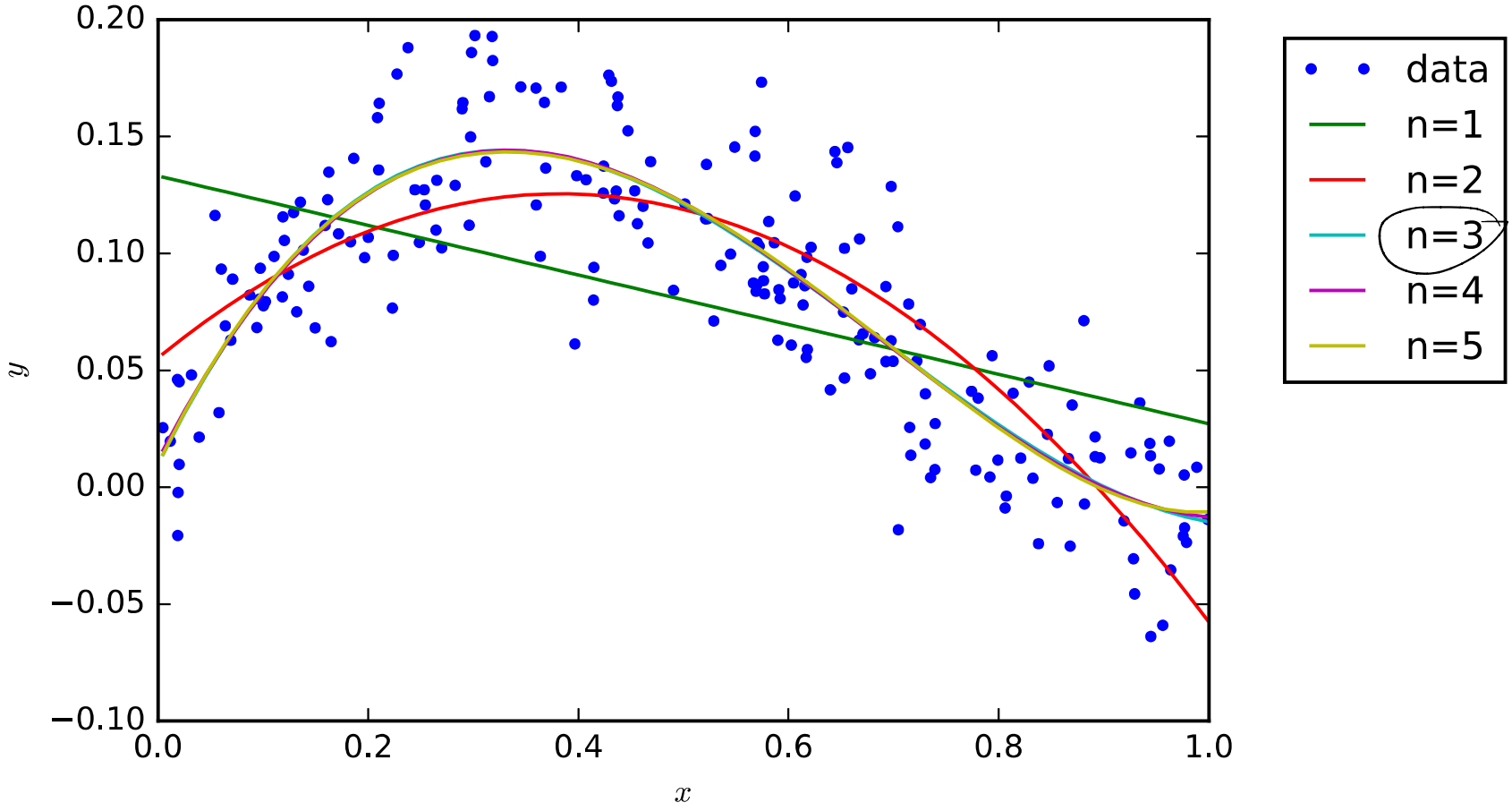
$y = ax + b$



Underfitting

$y = b + a x$

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



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.

pristine, unused data

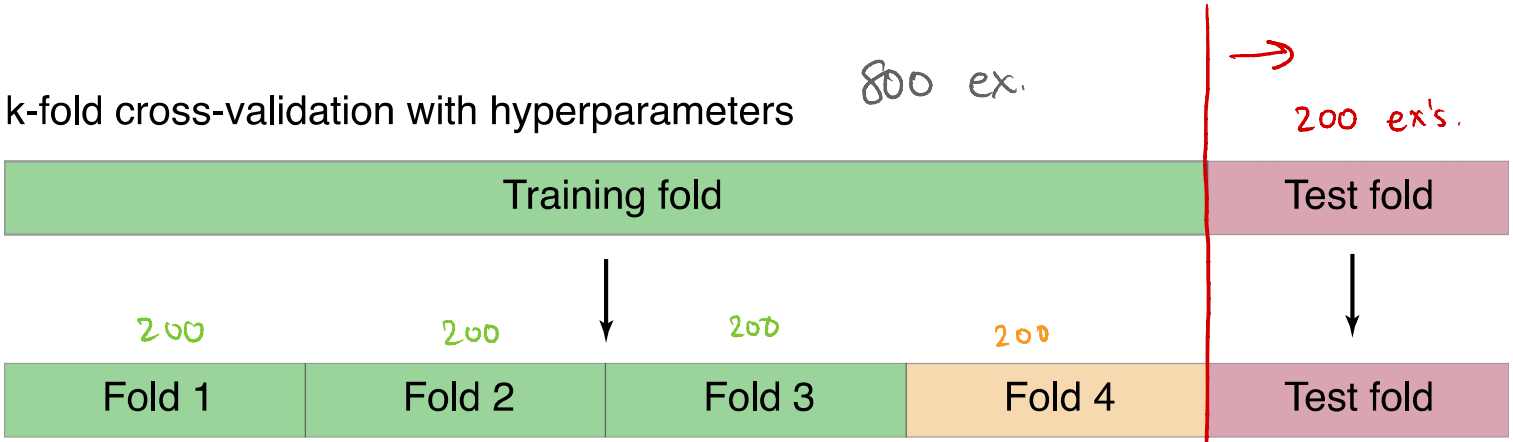
Evaluating generalization error

1000 ex's.

k-fold cross-validation with hyperparameters

800 ex.

200 ex's.



600 ex's for learning θ

4-fold cross validation

$n = 1, 2, 3, 4, 5$

$\mathcal{L}^1 \quad \mathcal{L}^2 \quad \mathcal{L}^3 \quad \mathcal{L}^4 \quad \mathcal{L}^5$

Evaluating generalization error

You can also change which fold is the validation fold to compute an average validation error or loss.

Evaluating generalization error

You can also change which fold is the validation fold to compute an average validation error or loss.

Evaluating generalization error

You can also change which fold is the validation fold to compute an average validation error or loss.

Fold 1	Fold 2	Fold 3	Val fold	Test fold
Fold 1	Fold 2	Val fold	Fold 4	Test fold
Fold 1	Val fold	Fold 3	Fold 4	Test fold
Val fold	Fold 2	Fold 3	Fold 4	Test fold

Training, validation, and testing

Explicit example:

10 classes

- The CIFAR-10 dataset we download will contain 60,000 images.
  - **Test set**: We will set aside 10,000 examples, never to be touched except one time at the end of training to **score our model.**

Training, validation, and testing

Explicit example:

- The CIFAR-10 dataset we download will contain 60,000 images.
  - **Test set**: We will set aside 10,000 examples, never to be touched except one time at the end of training to **score our model.**
- The remaining 50,000 images can be used for training and validation.
  - In **5-fold cross validation:**
    - We will split our data into **5 folds, each with 10,000 images.**
    - We will **train on 4 folds (40,000 images)** and evaluate on **1 fold (10,000 images)**. We will do this 5 times, each time changing which fold is the validation data.
    - We will average the validation accuracies (5 of them) to determine the best hyperparameters. → $n = 3$ is the best.
    - We can train one more time across all 50,000 images using the best hyperparameters.

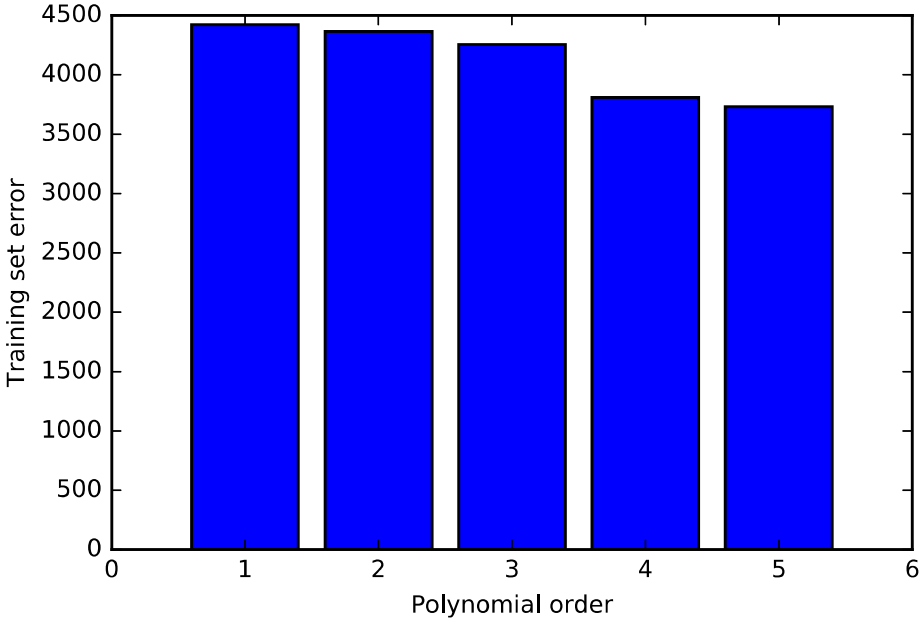
$$y = b + a_1 x + a_2 x^2 + a_3 x^3$$

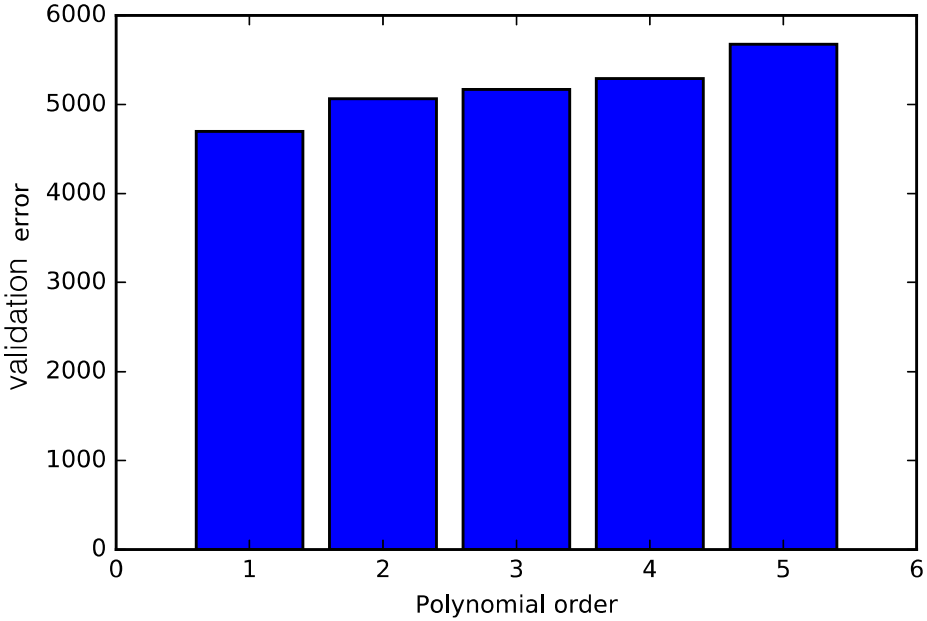
k-fold cross validation from housing example

Routines for training models and validating models.

```
def train_models(x, y):
    # Fit a polynomial model up to N=5
    N = 5
    thetas = []
    for i in np.arange(N):
        xhat = np.vstack((x, np.ones_like(x))) if i == 0 else np.vstack((x**(i+1), xhat))
        thetas.append(np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y)))

    return thetas

def val_models(x, y, thetas):
    N = 5
    errors = []
    for i in np.arange(N):
        xhat = np.vstack((x, np.ones_like(x))) if i == 0 else np.vstack((x**(i+1), xhat))
        errors.append(np.sqrt(np.sum((y - thetas[i].dot(xhat))**2) / len(y)))
    return errors
```

k-fold cross validation from housing example





Other types of optimization

We've talked about examples where we want to *minimize* a mean-square error or distance metric.

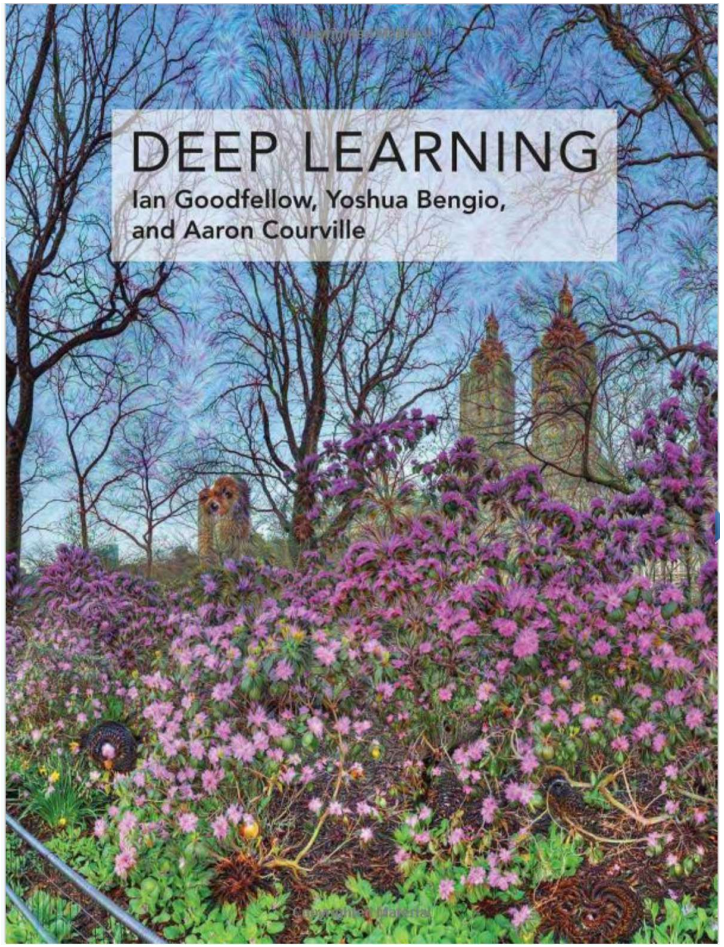
Another metric that we may want to maximize is the *probability of having observed the data*. In this framework, the data is modeled to have some distribution with parameters. We choose the parameters to maximize the probability of having observed our training data.

We will talk about this further when we discuss the "softmax classifier."

Reading

Reading:

Deep Learning, 5.7, 5.9, 5.10, 5.11.1.



Supervised classification, maximum-likelihood, and gradient descent

In this lecture, we'll cover some supervised learning techniques that are relevant for classification, and helpful for later lectures in neural networks.

Image classification

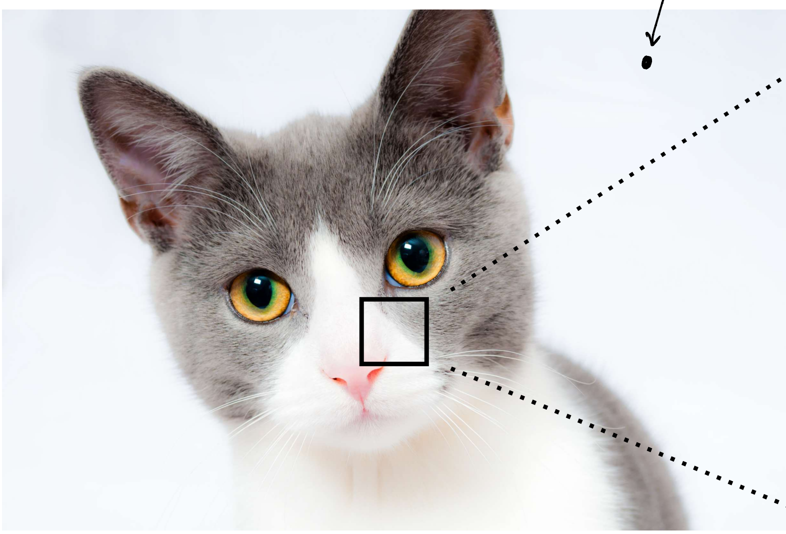
This image is licensed under CC0. https://www.pexels.com/photo/grey-and-white-short-fur-cat-104827/

Image classification

one pixel: R, G, B

[0, 255]

h



w

This image is licensed under CC0.

```
[[217, 215, 213, 213, 211, 210, 209, 207, 205, 203, 202, 201, 209],
 [218, 216, 215, 214, 211, 210, 209, 207, 205, 203, 202, 201, 209],
 [220, 218, 217, 216, 211, 210, 209, 207, 205, 203, 202, 201, 209],
 [220, 219, 217, 216, 211, 210, 209, 207, 205, 203, 202, 201, 209],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 218, 217, 216, 214, 212, 210, 209, 208, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214],
 [220, 220, 220, 220, 214, 214, 212, 210, 209, 207, 205, 205, 214]]
```

3D Tensor: w x h x 3

Images are stored in computers as a width x height x 3 array with numbers from [0, 255].

This is the input data for a computer vision algorithm.

Image classification

As the image is stored as an array of numbers, there are several challenges associated with image classification.

Image classification

Viewpoint variation:

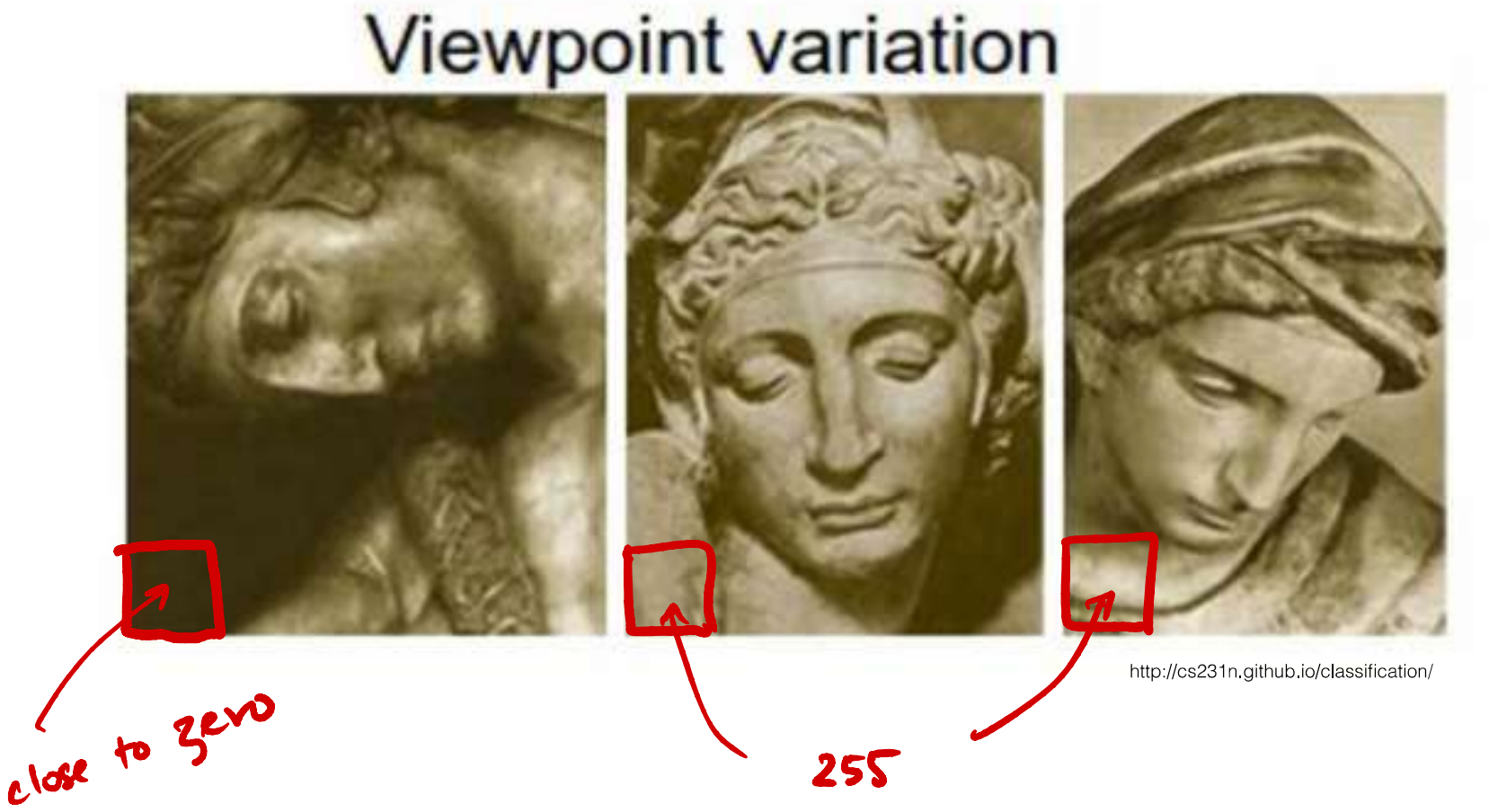


Image classification

Viewpoint variation

https://kevinzakka.github.io/2017/01/18/stn-part2/

Image classification

Illumination.

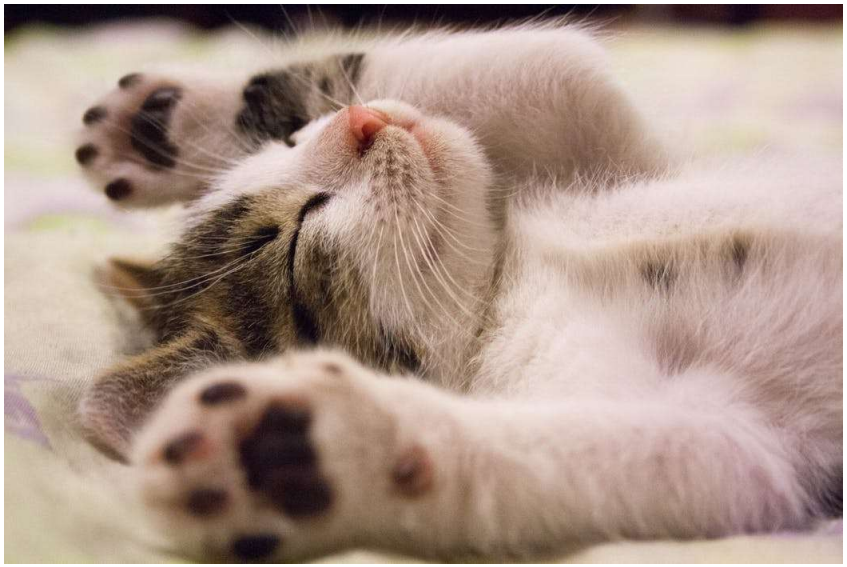
This image is licensed under CC0.

This image is licensed under CC0.

Image classification

Deformation.

This image is licensed under CC0.



This image is licensed under CC0.

This image is licensed under CC0.

Image classification

Occlusion.

This image is licensed under CC0.

This image is licensed under CC0.

This image is licensed under CC0.

Image classification

Background clutter.

This image is licensed under CC0.

This image is licensed under CC0.

Image classification

Intraclass variation.

Classifying image data?

When trying to classify an image, there are several approaches one could think of taking.

In this class, we use the data driven approach, where we let a machine learning algorithm see a lot of data and learn a function mapping the image to class.

TRAIN: data ⟶ model params, θ

(In deep NNs, these θ result in learning features of the data optimized to classify images.)

TEST: model ⟶ class

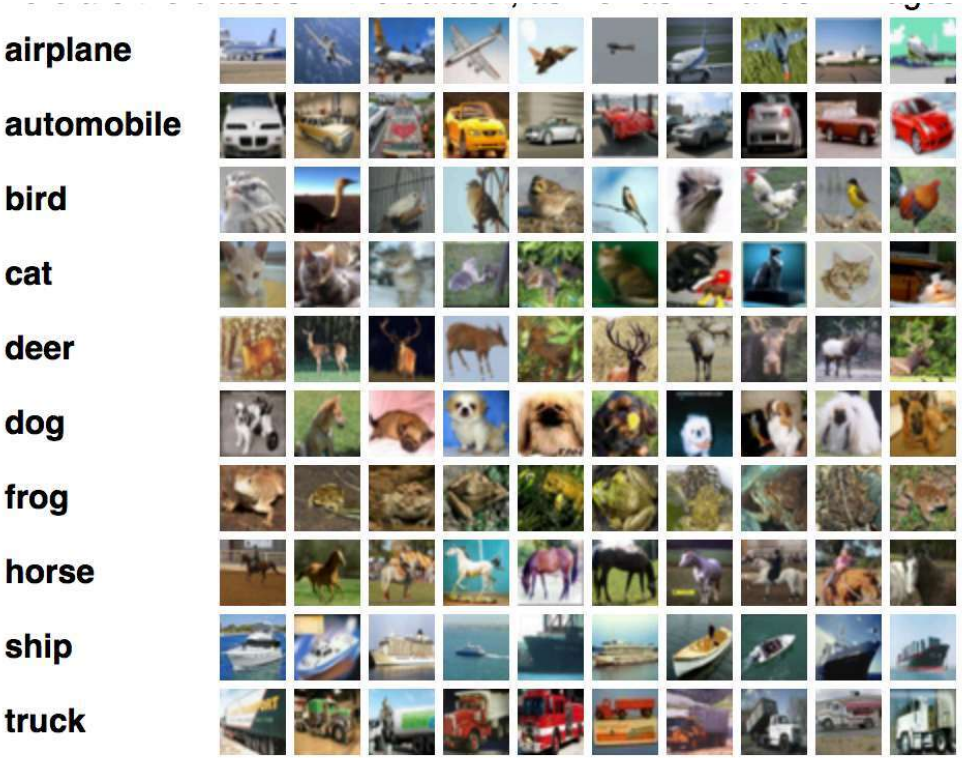
new data

Classifying image data

For data, we will use the CIFAR-10 dataset: http://www.cs.toronto.edu/~kriz/cifar.html

The dataset comprises:

- 60,000 images that are 32 x 32 pixels in color (hence 3 channels).
- 10 classes, with 6,000 images per class.
- 50,000 images are for training and 10,000 are for testing.



$32 \times 32 \times 3$

3072

$x^{(i)} \in \mathbb{R}$

$i = 1, 2, \ldots, 50000$

Classifying image data

```
# Loading CIFAR-10
import os
import pickle

def load_CIFAR10(ROOT):
  """ load all of cifar """
  xs = []
  ys = []
  for b in range(1,6):
    f = os.path.join(ROOT, 'data_batch_%d' % (b, ))
    X, Y = load_CIFAR_batch(f)
    xs.append(X)
    ys.append(Y)
  Xtr = np.concatenate(xs)
  Ytr = np.concatenate(ys)
  del X, Y
  Xte, Yte = load_CIFAR_batch(os.path.join(ROOT, 'test_batch'))
  return Xtr, Ytr, Xte, Yte

def load_CIFAR_batch(filename):
  """ load single batch of cifar """
  with open(filename, 'rb') as f:
    datadict = pickle.load(f)
    X = datadict['data']
    Y = datadict['labels']
    X = X.reshape(10000, 3, 32, 32).transpose(0,2,3,1).astype("float")
    Y = np.array(Y)
    return X, Y
```

Classifying image data

```
X_train, y_train, X_test, y_test = load_CIFAR10('cifar-10-batches-py')

print 'Training data shape: ', X_train.shape
print 'Training labels shape: ', y_train.shape
print 'Test data shape: ', X_test.shape
print 'Test labels shape: ', y_test.shape
```

```
Training data shape:  (50000, 32, 32, 3)
Training labels shape:  (50000,)
Test data shape:  (10000, 32, 32, 3)
Test labels shape:  (10000,)
```

(N, w, h, c)
(N,)

We will reshape these 32x32x3 arrays into a 3072-dimensional vector.

These constitute the “input” data.

Classifying image data?

$x^{(1)} \in \mathbb{R}^{3072}$ was automobile

$y^{(1)} = 1$

Supervised classification (cont.)

Consider a setup that is the same as the maximum likelihood classifier of the previous lecture. Imagine you were given a training set of input vectors, $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}\}$ and their corresponding classes, $\{y^{(1)}, y^{(2)}, \ldots, y^{(m)}\}$.

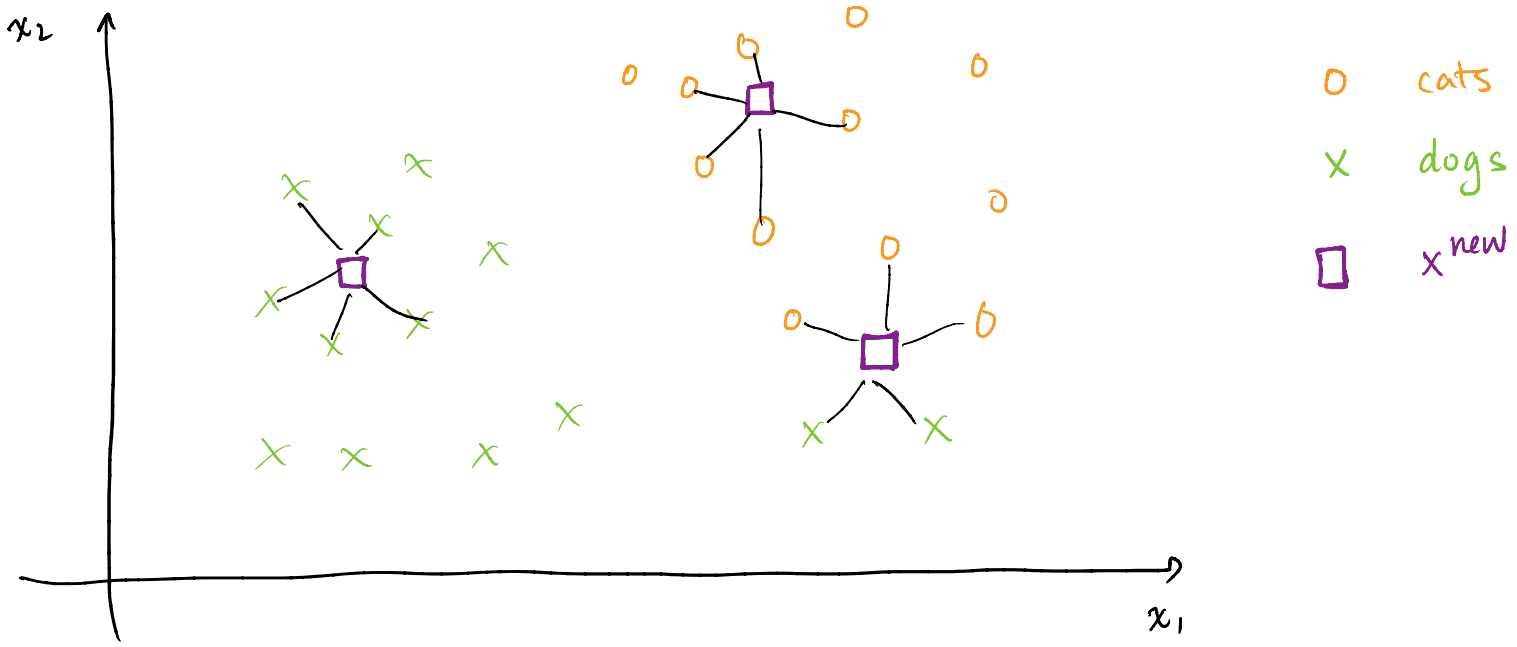
Now imagine that you were also given a new data point, $\mathbf{x}^{\text{new}}$. We found a way to classify through a probabilistic model, where we had to learn parameters, but isn't there a simpler way to classify that doesn't involve very much "machine learning" machinery?

k-nearest neighbors

$k = 5$

Intuitively, k-nearest neighbors says to find the k closest points (or nearest neighbors) in the training set, according to an appropriate metric. Each of its k nearest neighbors then vote according to what class it is in, and $\mathbf{x}^{\text{new}}$ is assigned to be the class with the most votes.

$x^{(i)} \in \mathbb{R}^2$



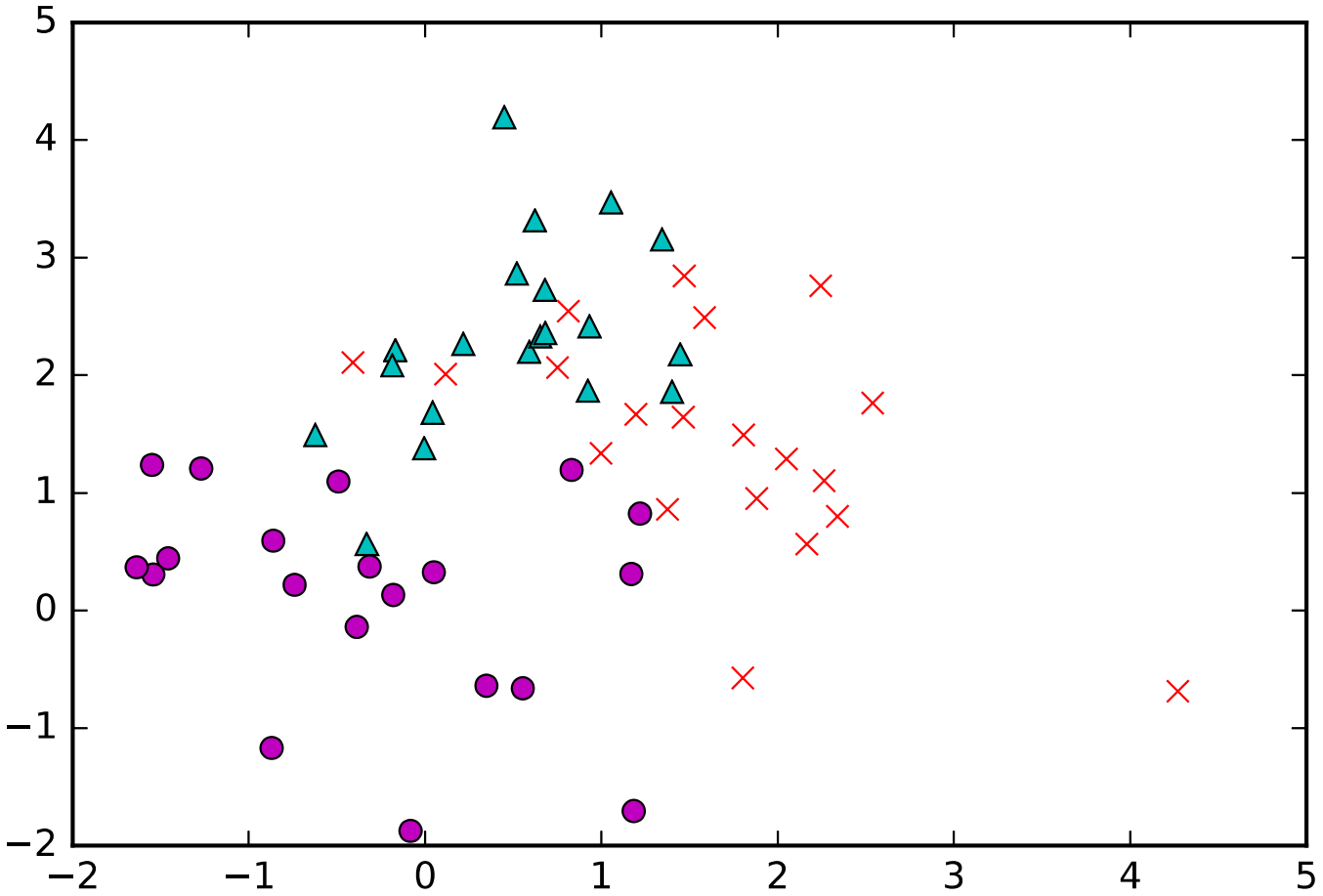
k-nearest neighbors

k-nearest neighbors, more formally

- Choose an appropriate distance metric, $d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$, returning the distance between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$. E.g., $d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \left\| \mathbf{x}^{(i)} - \mathbf{x}^{(j)} \right\|_2 = \sqrt{\sum_{k=1}^{n} \left(x_k^{(i)} - x_k^{(j)} \right)^2}$
- Choose the number of nearest neighbors, k.
- Take desired test data point, $\mathbf{x}^{\text{new}}$, and calculate $d(\mathbf{x}^{\text{new}}, \mathbf{x}^{(i)})$ for $i = 1, \ldots, m$.
- With $\{c_1, \ldots, c_k\}$ denoting the k indices corresponding to the k smallest $d(\mathbf{x}^{\text{new}}, \mathbf{x}^{(i)})$, classify $\mathbf{x}^{\text{new}}$ as the class that occurs most frequently amongst $\{y^{c_1}, \ldots, y^{c_k}\}$. If there is a tie, any of the tying classes may be selected.

k-nearest neighbors

Example data:



k-nearest neighbors

How do we train the classifier?

k-nearest neighbors

How do we train the classifier?

```
class KNearestNeighbor(object):

  def __init__(self):
    pass

  def train(self, X, y):
    self.X_train = X
    self.y_train = y
```

Pros? simple, fast

Cons? Memory intensive, b/c we need to store all the input data.

k-nearest neighbors

How do we test a new data point?

BROADCASTING

```
class KNearestNeighbor(object):

  def __init__(self):
    pass

  def train(self, X, y):
    self.X_train = X
    self.y_train = y

  def test(self, x_test, k=1):

    dists = np.linalg.norm(self.X_train.T - x_test, axis=1).T
    sortedIdxs = np.argsort(dists)
    closest_y = self.y_train[sortedIdxs[:k]]
    y_pred = np.argmax(np.bincount(closest_y));

    return y_pred
```

$X \in \mathbb{R}^2$

m examples

X_train : (2, m) array

X_test : (2,) array

(m, 2) (2,)

2

[ ]

bsxfun

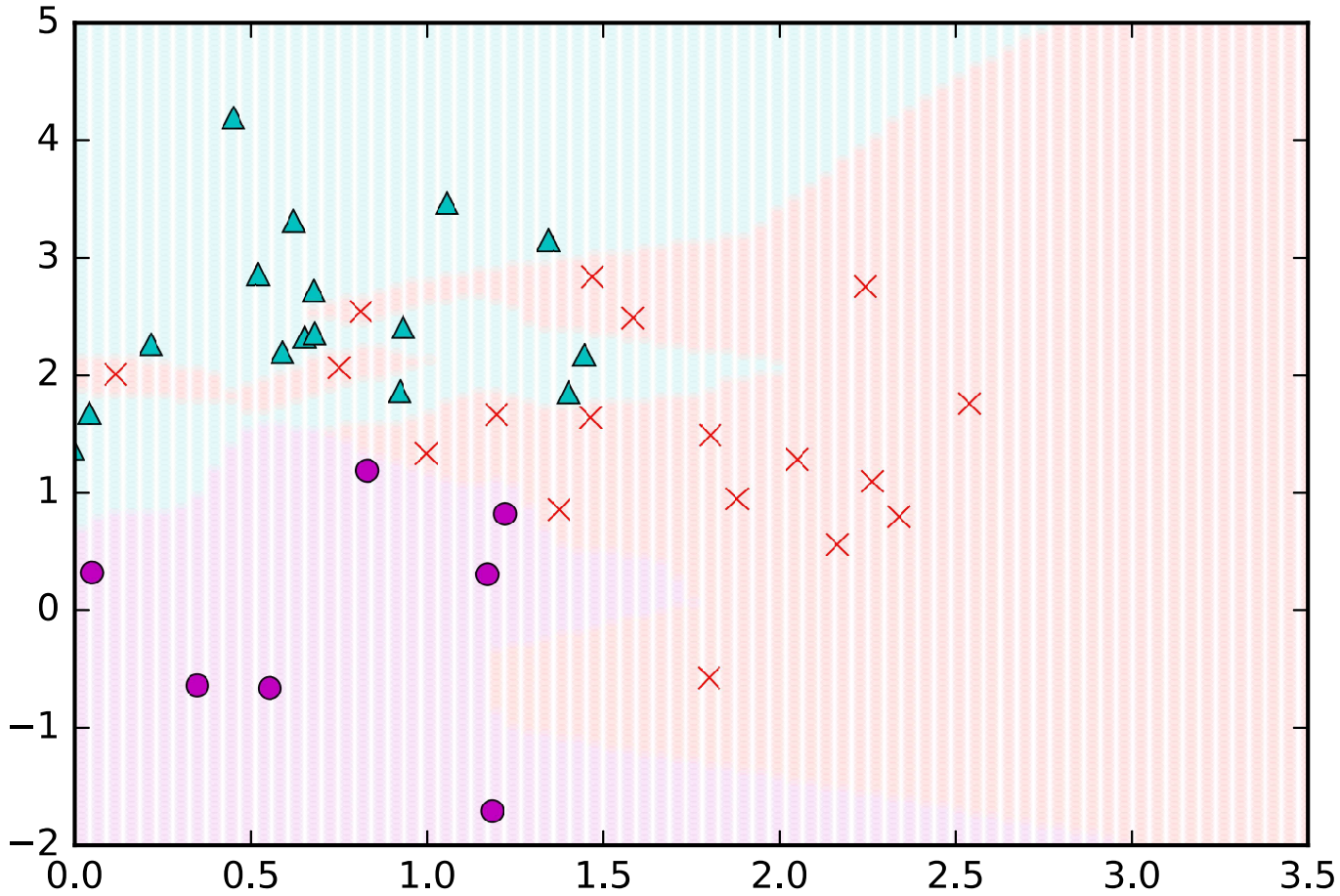
m

Pros? simple

Cons? slow, scale w/ the amount of training data

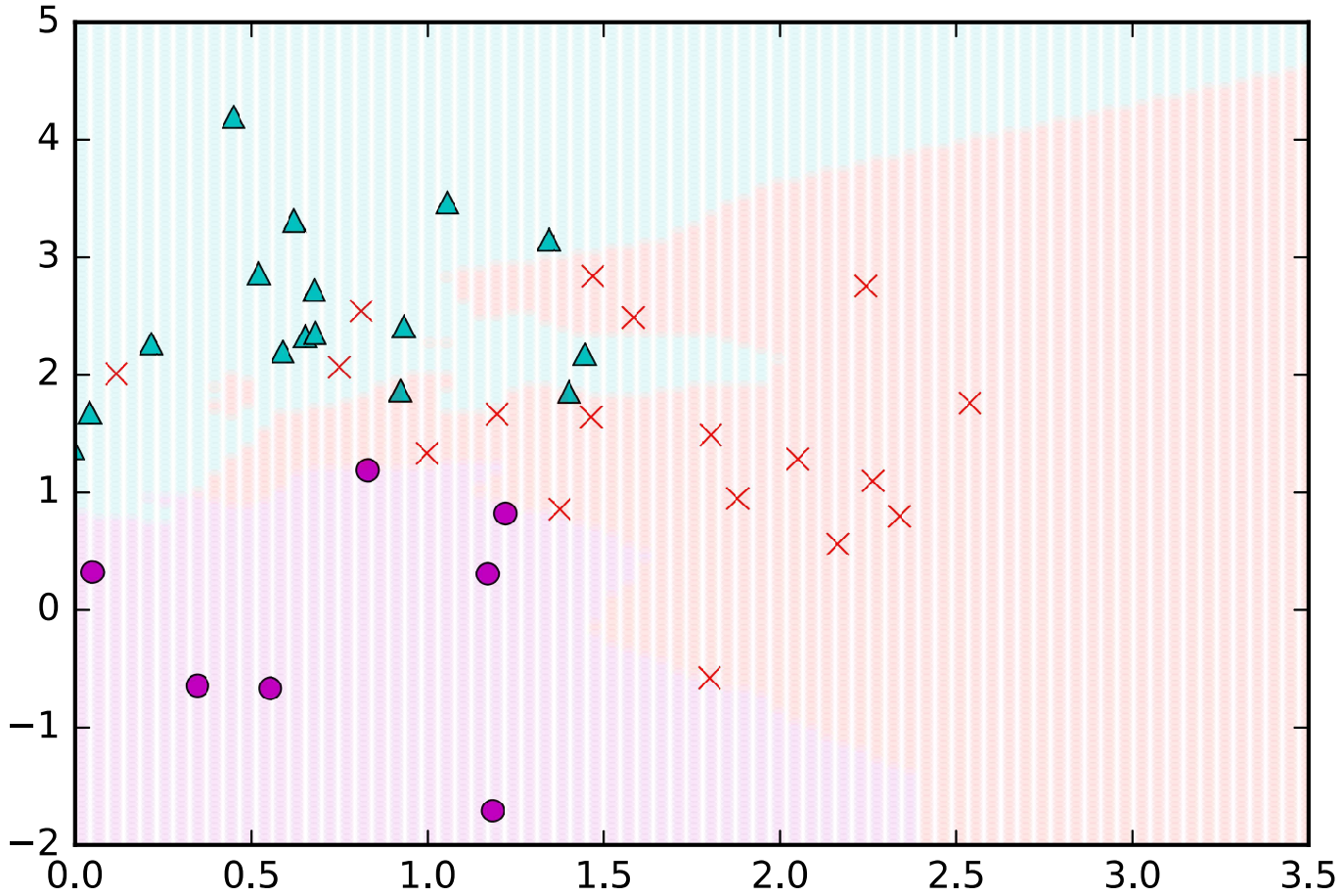
k-nearest neighbors

What a solution looks like for k=1 neighbor:



k-nearest neighbors

What a solution looks like for k=3 neighbors:



k-nearest neighbors

What are the hyperparameters of k-nearest neighbors?

k, distance metric

k-nearest neighbors

Why might k-nearest neighbors not be a good idea for image classification?

k-nearest neighbors

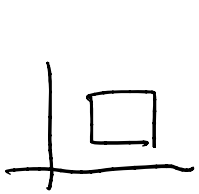
Why might k-nearest neighbors not be a good idea for image classification?

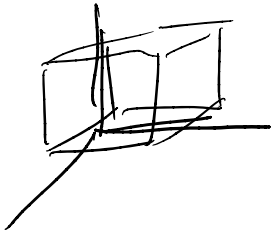
(all 3 images have same L2 distance to the one on the left)

Credit: CS231n, Stanford University

k-nearest neighbors

Why might k-nearest neighbors not be a good idea for image classification?





Curse of dimensionality:

- Images are very high-dimensional vectors, e.g., each CIFAR-10 image is a 3072 dimensional vector (and these are small images).

- Notions of “distance” become less intuitive in higher dimensions.
  - Distances in some dimensions matter more than others.
  - In higher-dimensional space, the volume increases exponentially.
  - This leaves a lot of empty space — and so the nearest neighbors may not be so near.