Lecture 2: Basics of machine learning

This lecture gives a basic introduction (or refresher depending on your background) on machine learning.

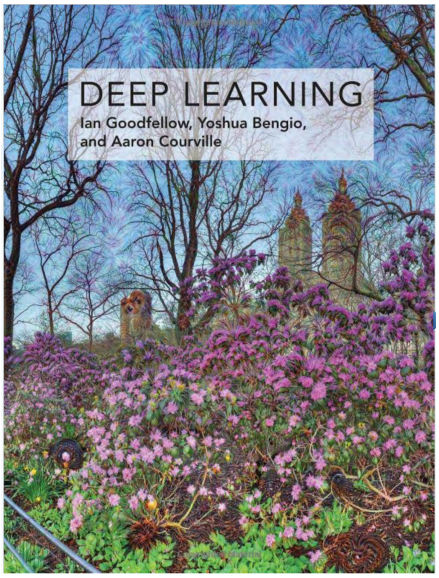
- Introduction to concepts in machine learning
- Cost functions
- Example: linear and polynomial regression
- Model complexity and overfitting
- Training set, validation set, test set
- Dealing with probabilistic cost functions and models
- Example: maximum-likelihood classification

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.

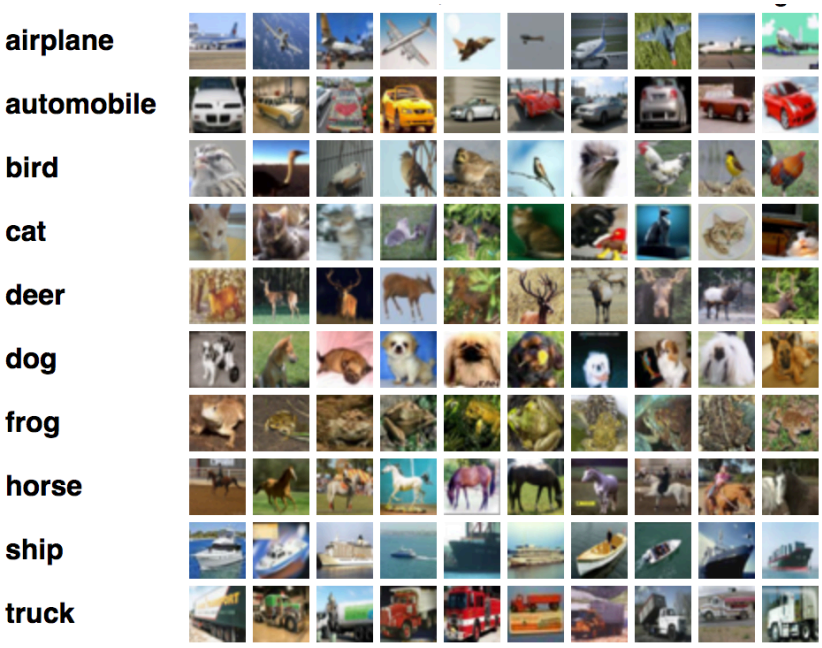


What is machine learning?

Machine learning uses statistical tools to estimate (or learn) functions, some of which may be fairly complex.

- Classification (discrete output): predicting the cateogry (one of k potential categories) given an input vector, $\mathbf{x} \in \mathbb{R}^n$. Example: classifying whether an image is of a cat or a dog.
- Regression (analog output): predicting the value given an input. Example: predicting housing prices from square footage.
- Synthesis and sampling: generating new examples that resemble the training data. Example: having a computer generate a spoken audio of a written text.
- Data imputation: filling in missing values of a vector. Example: Netflix predicting if you will like a show or movie.
- Denoising: taking a corrupt data sample $\mathbf{x}$ and outputting a cleaner sample $\mathbf{x}_{\text{clean}}$.
- And many others...

What is machine learning?



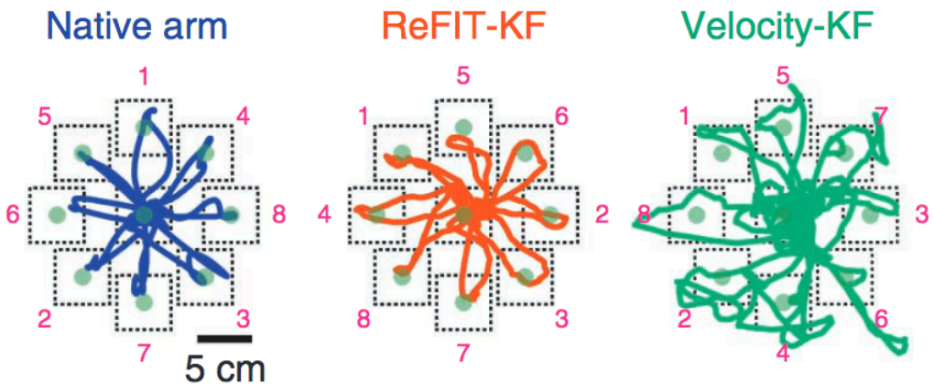
CIFAR-10 dataset, https://www.cs.toronto.edu/~kriz/cifar.html

What is machine learning?

Machine learning uses statistical tools to estimate (or learn) functions, some of which may be fairly complex.

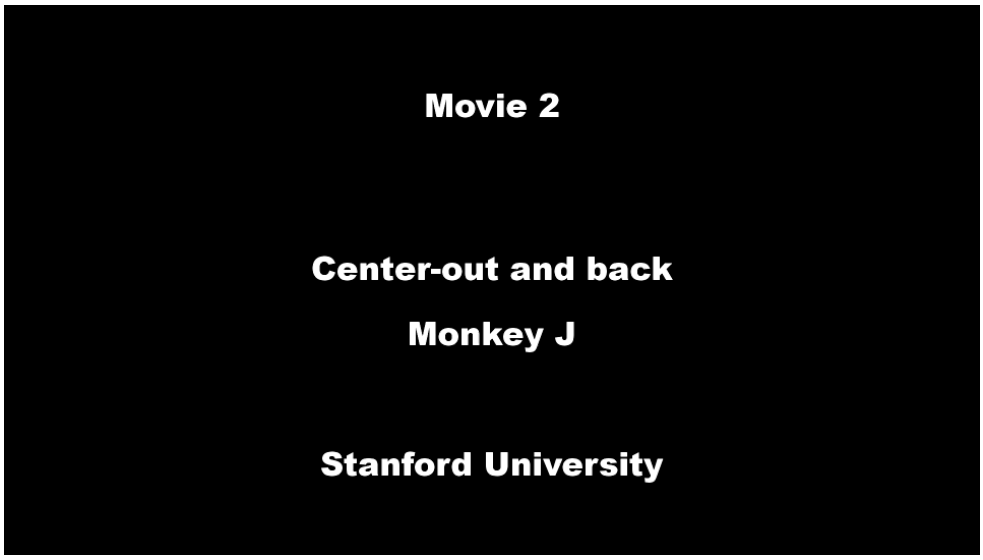
- Classification (discrete output): predicting the cateogry (one of k potential categories) given an input vector, $\mathbf{x} \in \mathbb{R}^n$. Example: classifying whether an image is of a cat or a dog.
- Regression (analog output): predicting the value given an input. Example: predicting housing prices from square footage.
- Synthesis and sampling: generating new examples that resemble the training data. Example: having a computer generate a spoken audio of a written text.
- Data imputation: filling in missing values of a vector. Example: Netflix predicting if you will like a show or movie.
- Denoising: taking a corrupt data sample $\mathbf{x}$ and outputting a cleaner sample $\mathbf{x}_{\text{clean}}$.
- And many others...

What is machine learning?



Gilja*, Nuyujukian*, et al., Nature Neuroscience 2012

What is machine learning?



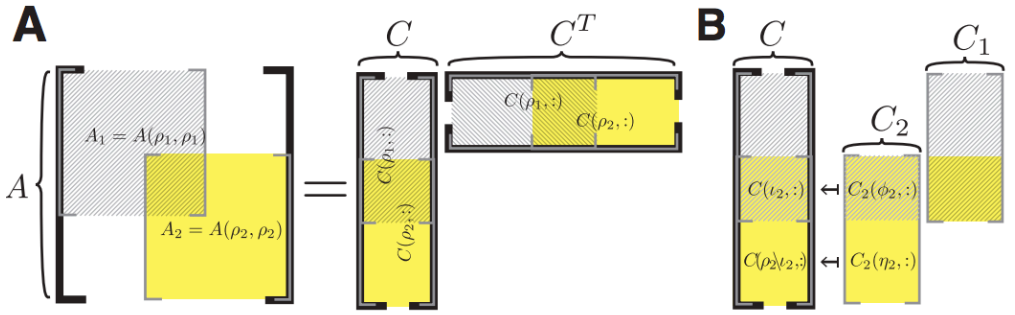
Gilja*, Nuyujukian*, … Kao, et al., Nature Neuroscience 2012

What is machine learning?

Machine learning uses statistical tools to estimate (or learn) functions, some of which may be fairly complex.

- Classification (discrete output): predicting the cateogry (one of k potential categories) given an input vector, $\mathbf{x} \in \mathbb{R}^n$. Example: classifying whether an image is of a cat or a dog.
- Regression (analog output): predicting the value given an input. Example: predicting housing prices from square footage.
- Synthesis and sampling: generating new examples that resemble the training data. Example: having a computer generate a spoken audio of a written text.
- Data imputation: filling in missing values of a vector. Example: Netflix predicting if you will like a show or movie.
- Denoising: taking a corrupt data sample $\mathbf{x}$ and outputting a cleaner sample $\mathbf{x}_{\text{clean}}$.
- And many others...

What is machine learning?



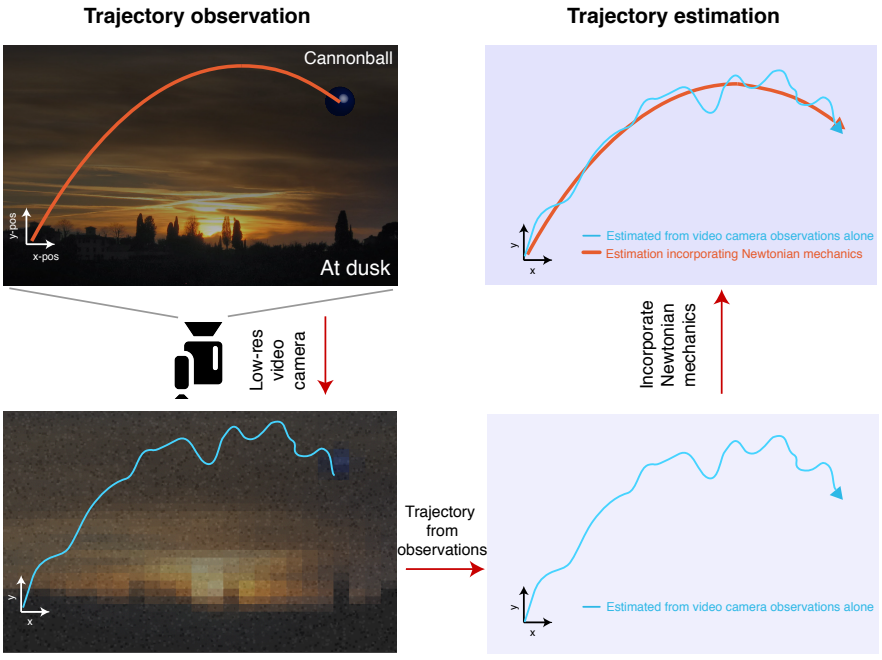
Bishop & Yu, NIPS 2014

What is machine learning?

Machine learning uses statistical tools to estimate (or learn) functions, some of which may be fairly complex.

- Classification (discrete output): predicting the cateogry (one of k potential categories) given an input vector, $\mathbf{x} \in \mathbb{R}^n$. Example: classifying whether an image is of a cat or a dog.
- Regression (analog output): predicting the value given an input. Example: predicting housing prices from square footage.
- Synthesis and sampling: generating new examples that resemble the training data. Example: having a computer generate a spoken audio of a written text.
- Data imputation: filling in missing values of a vector. Example: Netflix predicting if you will like a show or movie.
- Denoising: taking a corrupt data sample $\mathbf{x}$ and outputting a cleaner sample $\mathbf{x}_{\text{clean}}$.
- And many others...

What is machine learning?



Kao et al., Nature Communications 2015

What is machine learning?

Machine learning uses statistical tools to estimate (or learn) functions, some of which may be fairly complex.

- Classification (discrete output): predicting the cateogry (one of k potential categories) given an input vector, $\mathbf{x} \in \mathbb{R}^n$. Example: classifying whether an image is of a cat or a dog.
- Regression (analog output): predicting the value given an input. Example: predicting housing prices from square footage.
- Synthesis and sampling: generating new examples that resemble the training data. Example: having a computer generate a spoken audio of a written text.
- Data imputation: filling in missing values of a vector. Example: Netflix predicting if you will like a show or movie.
- Denoising: taking a corrupt data sample $\mathbf{x}$ and outputting a cleaner sample $\mathbf{x}_{\text{clean}}$.
- And many others...

Types of machine learning problems

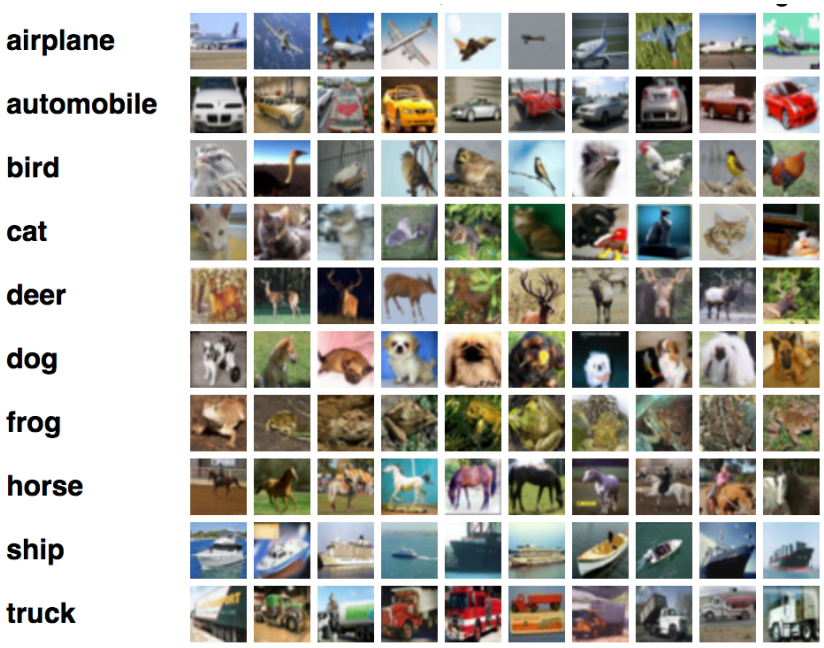
- *Supervised learning* involves applications where input vectors, $\mathbf{x}$, and their target vectors, $\mathbf{y}$, are known. The goal is to learn a function f that predicts $\mathbf{y}$ given $\mathbf{x}$, i.e., $\mathbf{y} = f(\mathbf{x})$.

- *Unsupervised learning* involves discovering structure in input vectors, absent of knowledge of their target vectors. Examples include finding similar input vectors (clustering), distributions of the inputs (density estimation) or dimensionality reduction (visualization).

- *Reinforcement learning* involves finding suitable actions in certain scenarios to maximize a given reward. It discovers policies through trial and error.

- This class will be primarily dealing with problems in supervised learning. We will discuss unsupervised and reinforcement learning at a higher level.

Classification



CIFAR-10 dataset, https://www.cs.toronto.edu/~kriz/cifar.html

Types of machine learning problems

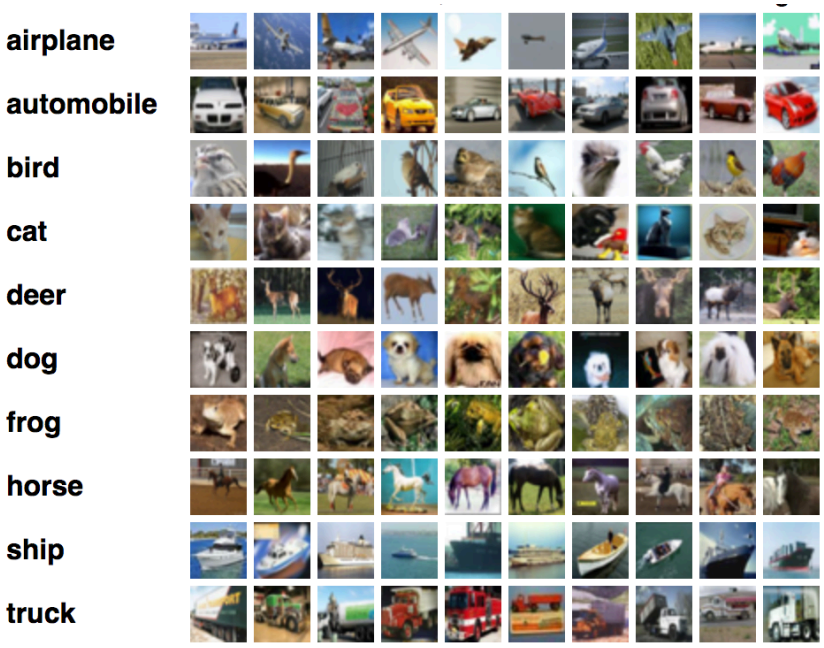
- *Supervised learning* involves applications where input vectors, $\mathbf{x}$, and their target vectors, $\mathbf{y}$, are known. The goal is to learn a function f that predicts $\mathbf{y}$ given $\mathbf{x}$, i.e., $\mathbf{y} = f(\mathbf{x})$.

- *Unsupervised learning* involves discovering structure in input vectors, absent of knowledge of their target vectors. Examples include finding similar input vectors (clustering), distributions of the inputs (density estimation) or dimensionality reduction (visualization).

- *Reinforcement learning* involves finding suitable actions in certain scenarios to maximize a given reward. It discovers policies through trial and error.

- This class will be primarily dealing with problems in supervised learning. We will discuss unsupervised and reinforcement learning at a higher level.

Classification



CIFAR-10 dataset, https://www.cs.toronto.edu/~kriz/cifar.html

Types of machine learning problems

- *Supervised learning* involves applications where input vectors, $\mathbf{x}$, and their target vectors, $\mathbf{y}$, are known. The goal is to learn a function f that predicts $\mathbf{y}$ given $\mathbf{x}$, i.e., $\mathbf{y} = f(\mathbf{x})$.
- *Unsupervised learning* involves discovering structure in input vectors, absent of knowledge of their target vectors. Examples include finding similar input vectors (clustering), distributions of the inputs (density estimation) or dimensionality reduction (visualization).
- *Reinforcement learning* involves finding suitable actions in certain scenarios to maximize a given reward. It discovers policies through trial and error.
- This class will be primarily dealing with problems in supervised learning. We will discuss unsupervised and reinforcement learning at a higher level.

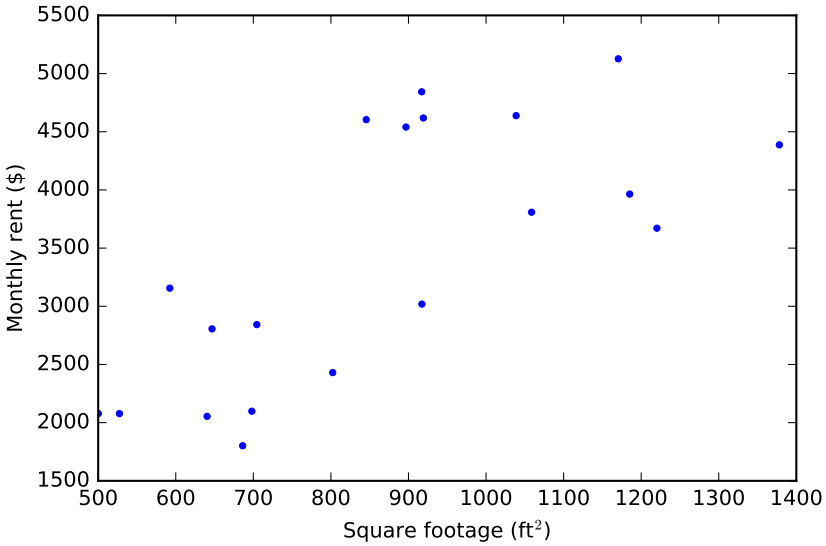
Focus of this class:

This class will focus on **supervised** learning problems.

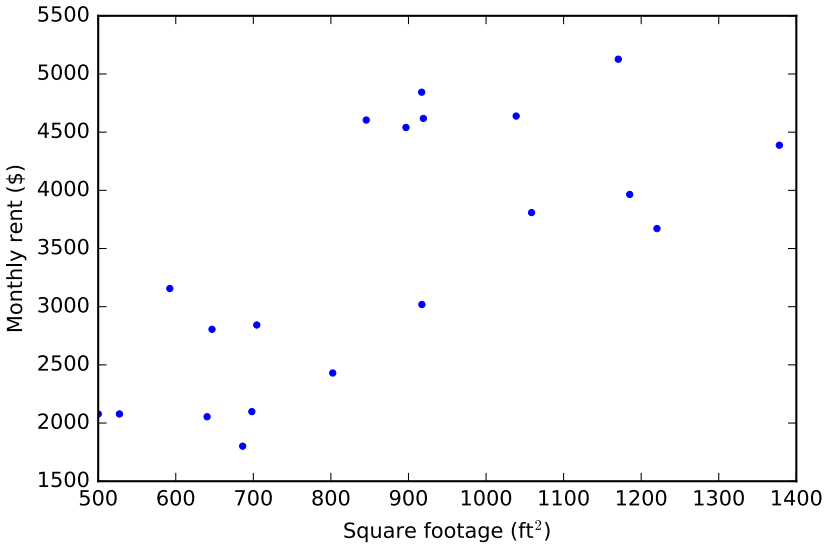
This class will also focus on **classification** largely (e.g., when considering convolutional neural networks) as well as some instances of **regression** (e.g., when considering recurrent neural networks)

An example of supervised learning

Let's say we want to rent a home in Westwood, and we wanted to know if we were getting a good deal. **(Warning: this data is synthetic! Scrape the real data if you're curious.)**



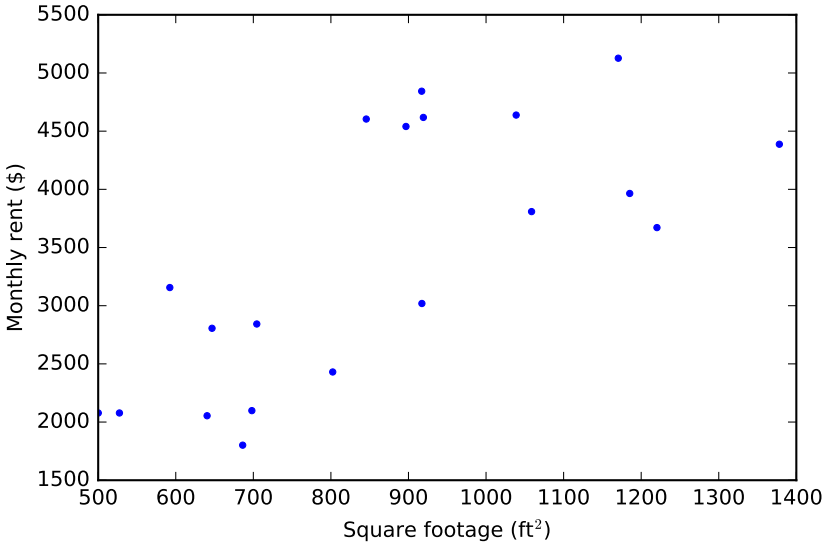
An example of supervised learning



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



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:

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

- Cost function:

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

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:

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

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

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:

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

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:

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

Aside: vector and matrix derivatives

Example: derivative with respect to a vector

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

Aside: vector and matrix derivatives

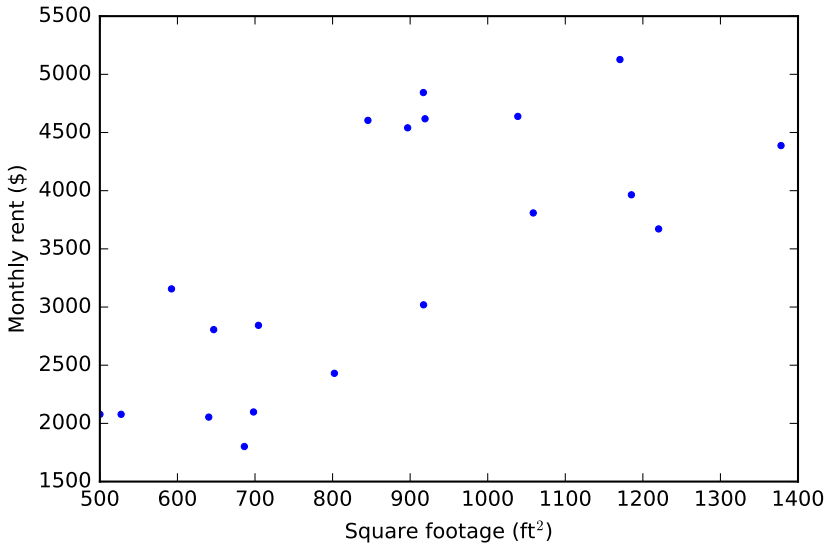
Example: derivative with respect to a vector

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

Aside: vector and matrix derivatives

Matrix derivatives

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:

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

Back to our supervised learning example

Let's now take derivatives:

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

Back to our supervised learning example

Back to our supervised learning example

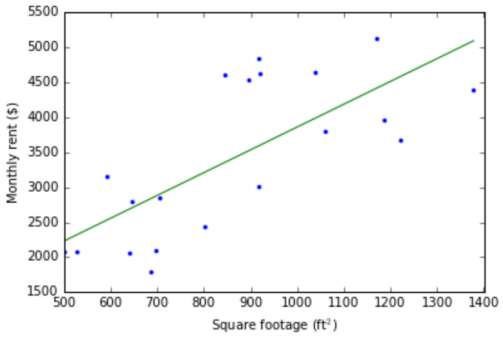
This solution is called least-squares, and it appears in a variety of linear applications.

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

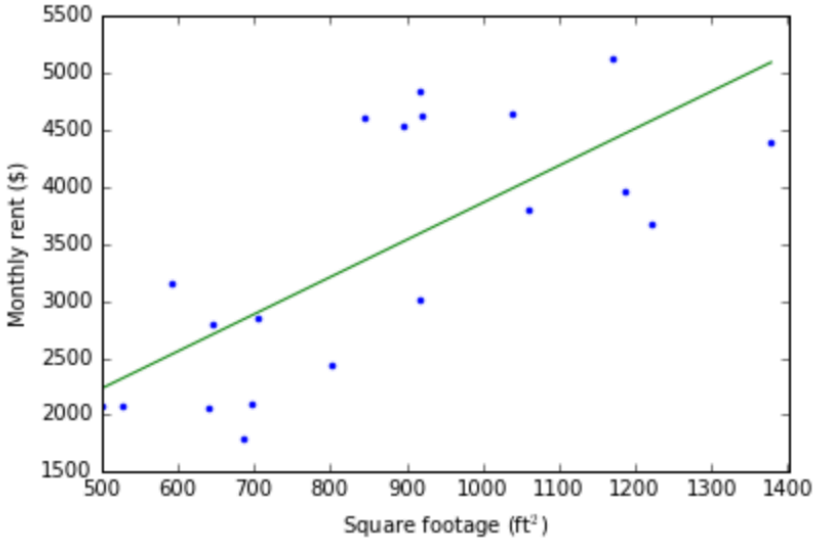
Back to our supervised learning example

```
# 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?

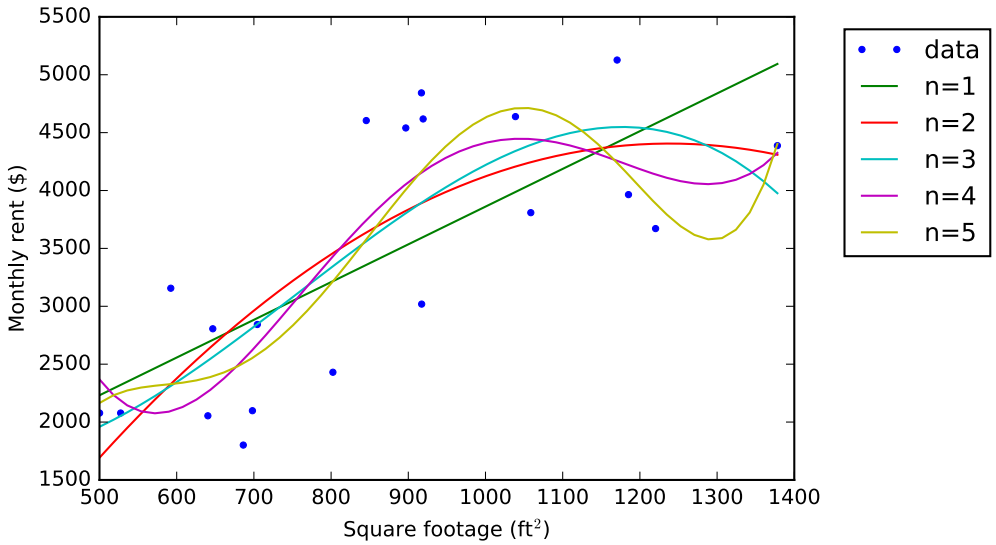


More generally…

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

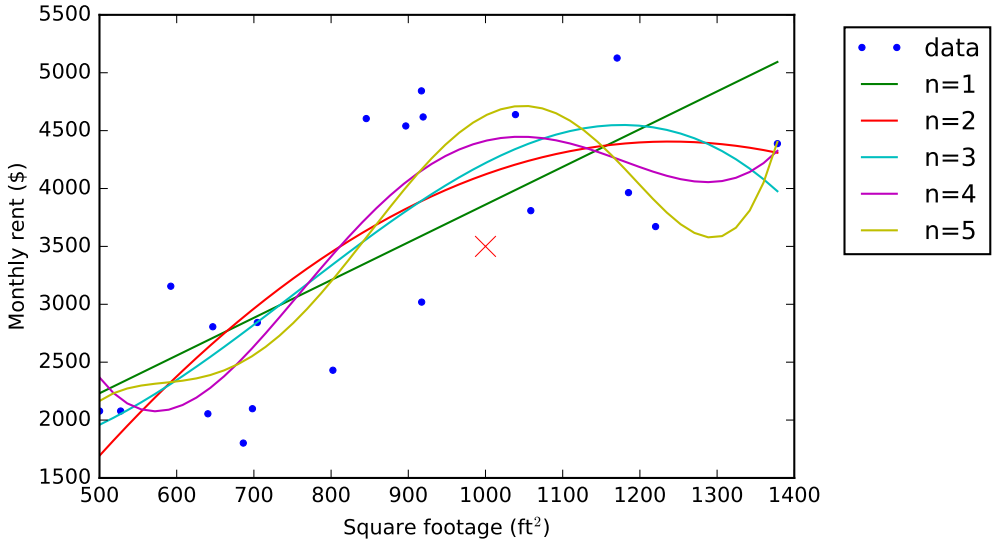
$$y = b + a_1 x_1 + a_2 x^2 + \cdots + a_n x^n$$

More generally…



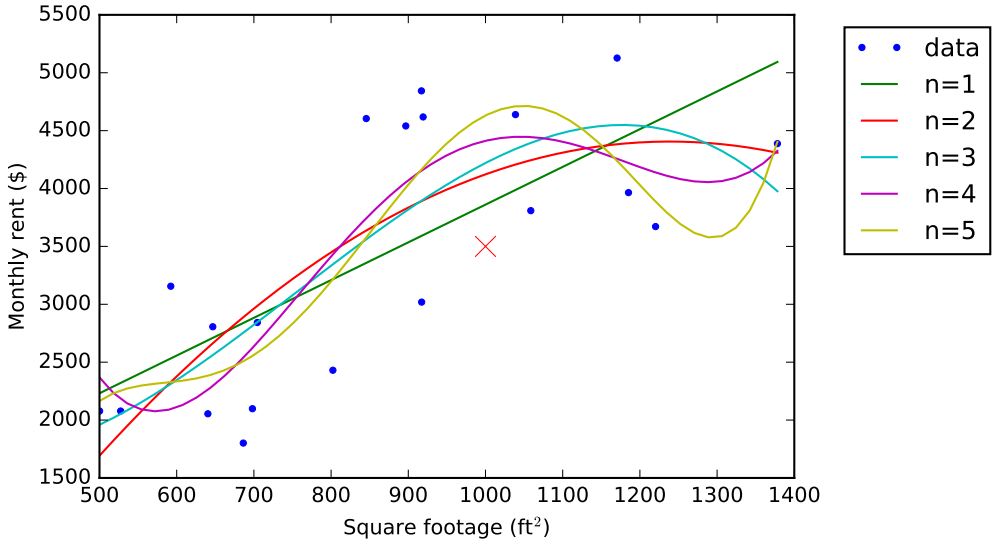
Now, a higher degree polynomial will *always* fit the **provided** data better. (Why)?

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

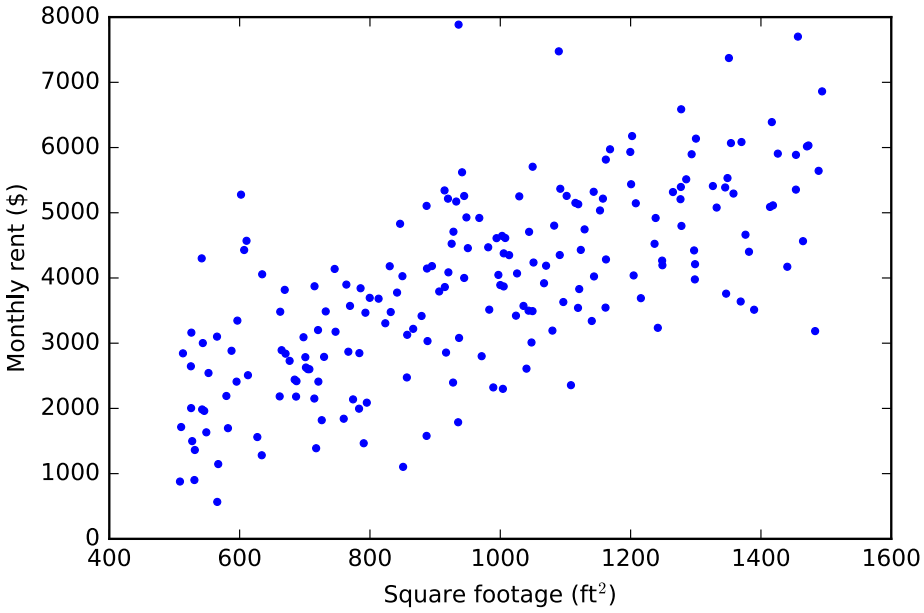
This idea of generalization can be made more formal by introducing the concepts of a *training set* and *testing set*.

- **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.

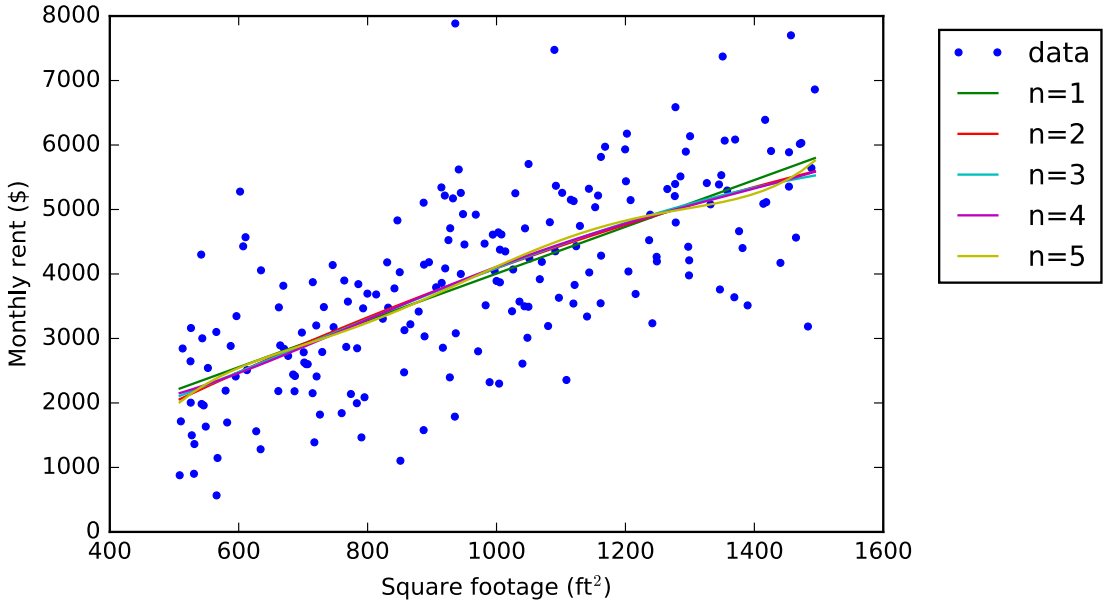
There is also a notion of validation data, which we will get to later.

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

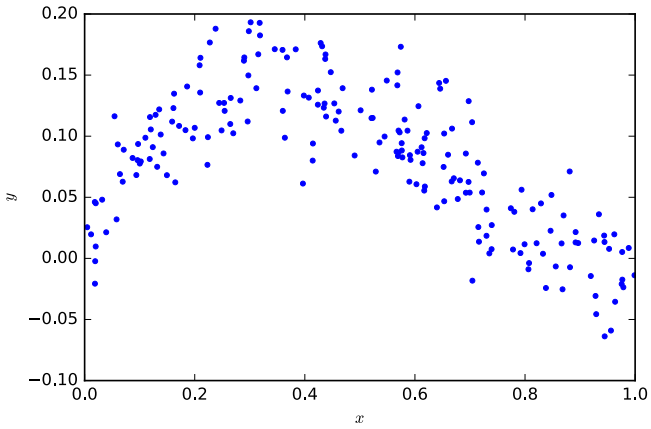
This suggests that when *a lot* of data is available, it may be appropriate to use more complex models. (Another technique we will discuss later, regularization, also helps with overfitting.) This is a shadow of things to come (i.e., neural networks which have large complexity).

Underfitting

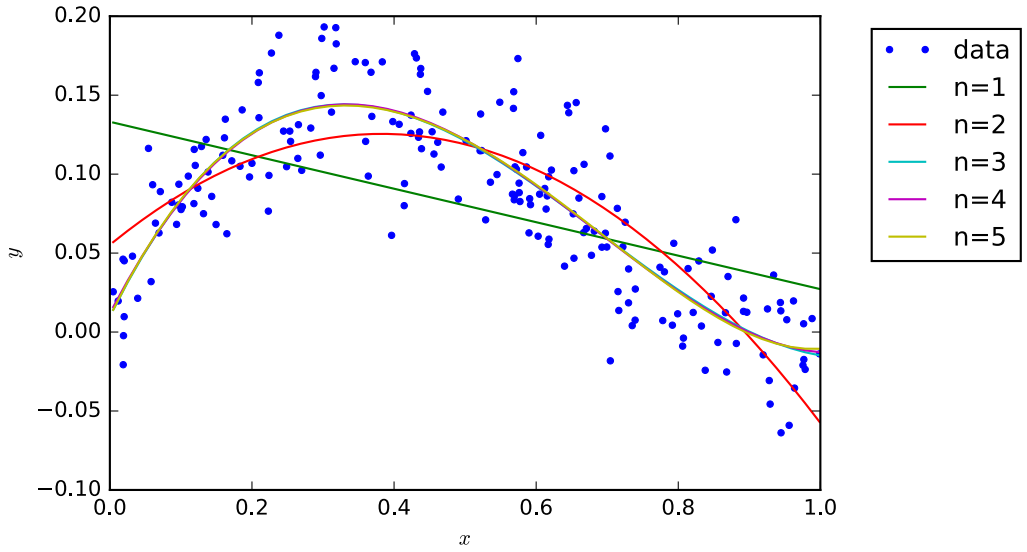
At the same time, we can not make the model *overly simple* as then we will underfit the data. This corresponds to a model that has both high training and high testing error, and the fit generally will not improve with more training data because the model is not expressive enough.

Underfitting

```
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



How do we think of overfitting and underfitting?

Overfitting and underfitting are important problems to address in machine learning.

A more rigorous framework to think of these is as the *bias* and *variance* of the estimate of our parameters.

Estimators

What is an estimator?

- Assume, for now, that there is a set of parameters given by θ.
- The estimator, $\hat{\theta}$, is a single "best" estimate of θ. (Note that we are not estimating the *distribution* of θ. We are treating θ as taking on a single value.)
- Given a training set of m iid examples, $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}$, the estimator may be generally defined as:

$$\hat{\theta}_m = g(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)})$$

- $\hat{\theta}_m$ ought be treated as a random variable, as we achieve an estimate of it through the random samples we have as training data.

Estimator bias

The *bias* of an estimator, $\hat{\theta}_m$, is defined as:

$$\text{bias}(\hat{\theta}_m) = \mathbb{E}(\hat{\theta}_m) - \theta$$

Note:

- The expectation is taken over the data (where the randomness is introduced through samples $\mathbf{x}^{(i)}$).
- Informally, the bias measures how close $\hat{\theta}_m$ comes to estimating θ on average.
- An estimator is called *unbiased* if $\text{bias}(\hat{\theta}_m) = 0$.

Estimator bias example

Estimator bias (example)

- Let $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ be iid samples from a Bernoulli distribution with mean θ.
- The *sample mean* estimator is given by:

$$\hat{\theta}_m = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

Intuitively, this ought to be a very reasonable estimator of the Bernoulli mean (or equivalently parameter).

Estimator bias example

Then, the bias of this estimator is:

$$\text{bias}(\hat{\theta}_m) = \mathbb{E}\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right) - \theta$$

Estimator variance

Estimator variance

An unbiased estimator may on average estimate θ, but any single instance of $\hat{\theta}$ may deviate from θ. The variance of $\hat{\theta}$ in predicting is called the variance of the estimator, denoted $\text{var}(\hat{\theta})$.

Some notes:

- This variability is due to the data samples that you get.
- The *standard error* of the estimator is $\sqrt{\text{var}(\hat{\theta})}$ and is commonly denoted $\text{SE}(\hat{\theta})$.

Estimator variance (example)

Let $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ be iid samples from a Bernoulli distribution with mean θ. The *sample mean* estimator is given by:

$$\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

Then, the variance of this estimator is:

$$\mathrm{var}(\hat{\theta}_m) \quad = \quad \mathrm{var}\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right)$$

Estimator variance (example)

A common metric used in evaluating the average of the data is the standard error of the mean. Given iid samples $x^{(1)}, \ldots, x^{(m)}$, the standard error of the mean is given by:

$$\begin{aligned}
\mathrm{SE}(\hat{\mu}_m) &= \sqrt{\mathrm{var}\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right)} \\
&= \sqrt{\frac{1}{m^2}\sum_{i=1}^{m} \mathrm{var}(x^{(i)})} \\
&= \sqrt{\frac{1}{m^2} m \mathrm{var}(x^{(i)})} \\
&= \frac{\sigma}{\sqrt{m}}
\end{aligned}$$

What does this mean intuitively?

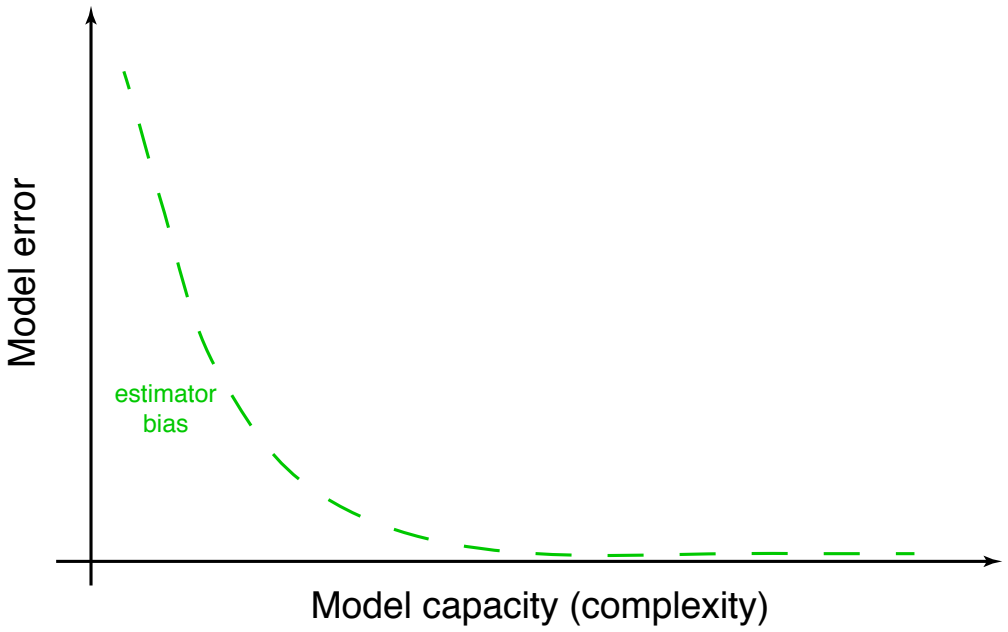
Trading off bias and variance

We may at times have to choose between a low-bias, high-variance estimator, or a high-bias, low-variance estimator.

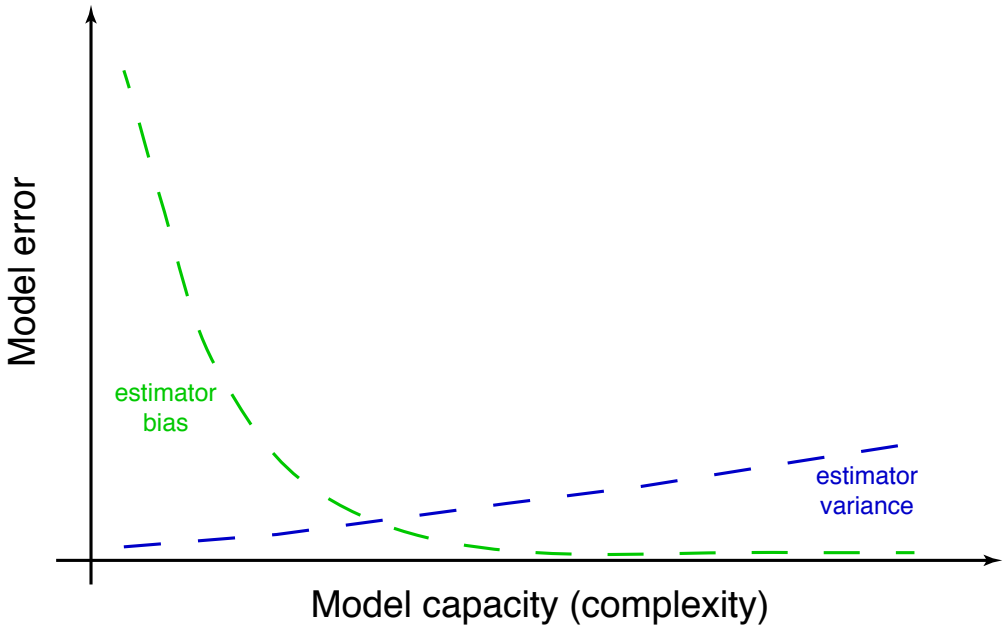
- The mean squared error (MSE) naturally trades these two off.

$$\begin{aligned}
\text{MSE} &= \mathbb{E}(\hat{\theta}_m - \theta)^2 \\
&= \mathbb{E}(\hat{\theta}_m - \mathbb{E}\hat{\theta}_m + \mathbb{E}\hat{\theta}_m - \theta)^2 \\
&= \mathbb{E}(\hat{\theta}_m - \mathbb{E}\hat{\theta}_m)^2 + \mathbb{E}\left[2(\hat{\theta}_m - \mathbb{E}\hat{\theta}_m)(\mathbb{E}\hat{\theta}_m - \theta)\right] + \mathbb{E}(\mathbb{E}\hat{\theta}_m - \theta)^2 \\
&= \text{var}(\hat{\theta}_m) + 2\text{bias}(\hat{\theta}_m)\mathbb{E}(\hat{\theta}_m - \mathbb{E}\hat{\theta}_m) + \mathbb{E}\text{bias}(\hat{\theta}_m)^2 \\
&= \text{var}(\hat{\theta}_m) + \text{bias}(\hat{\theta}_m)^2
\end{aligned}$$

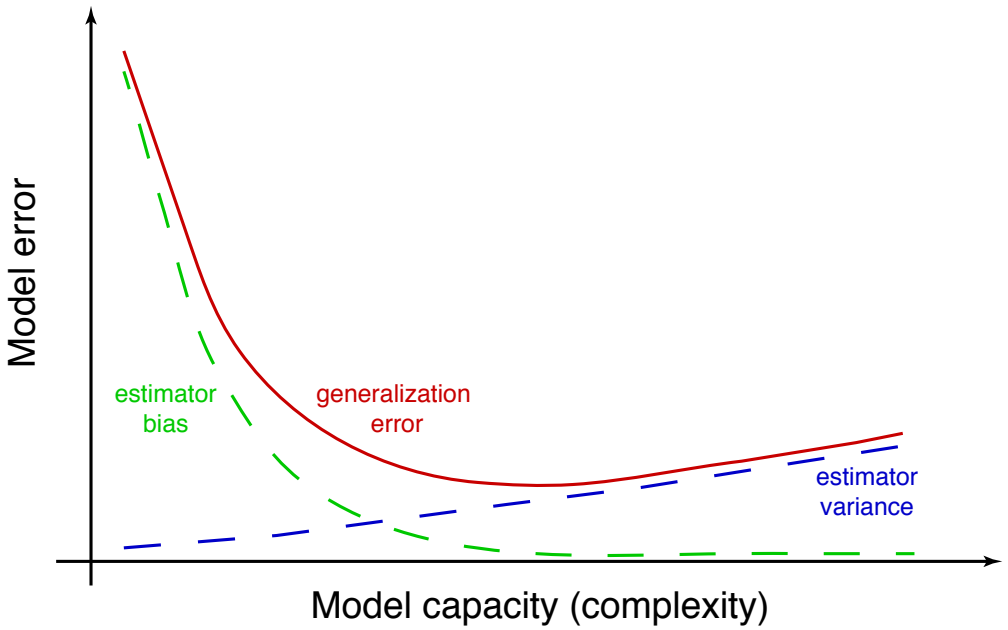
- Thus, minimizing MSE will simultaneously minimize the bias and variance of the estimator.



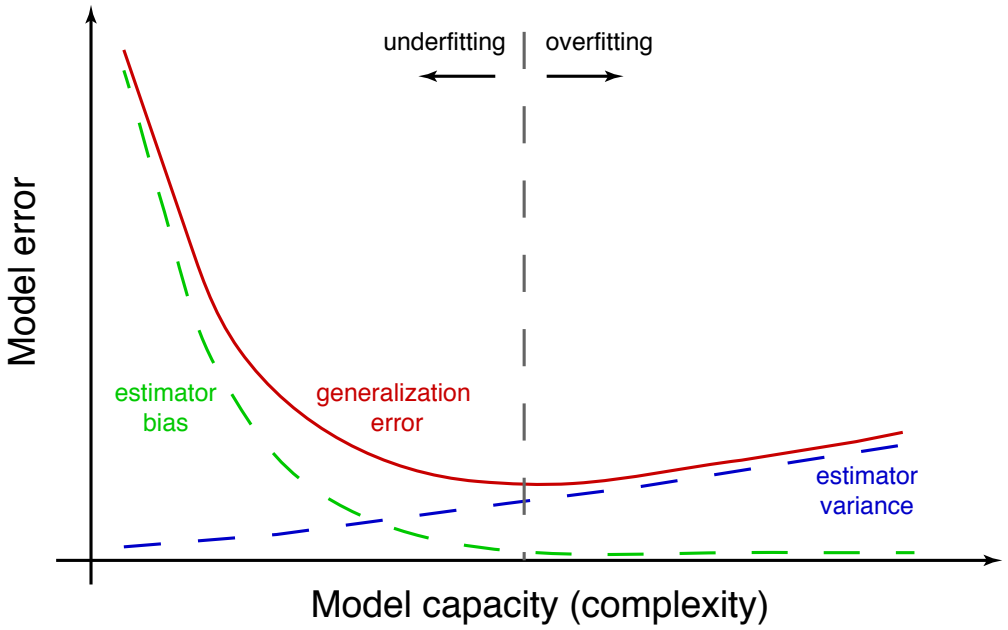
A picture to then have in mind



A picture to then have in mind



A picture to then have in mind



Picking a best model

From this picture, a reasonable way to pick a model is to assess its generalization error, and pick a setting of the parameters that results in minimal value.

A brief aside:

Sometimes there may be scenarios where dataset size is so limited that it is not possible to have a reasonably good test dataset. In these cases, we may utilize *model selection*, which is itself an entire field.

At a high level, the principles of model selection is to find a way to *penalize* the model for being overly complex. There is an art to designing some penalty terms.

Criterion for selecting models include:

- Bayes information criterion
- Akaike information criterion
- Deviance information criterion

My recommendation: if you have enough data to make a test set, do it.

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.

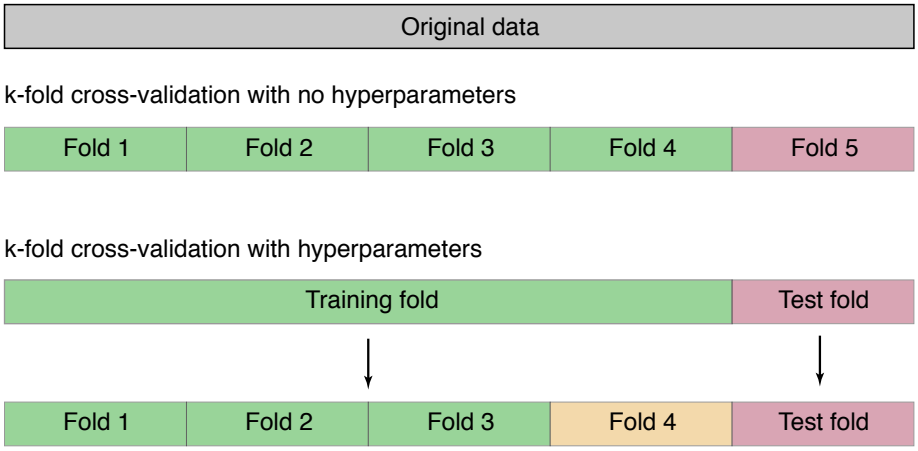
Note, in many cases, testing and validation datasets are used interchangeably as datasets that are used to evaluate the model. In this scenario, hyperparameters would also be optimized using training data.

k-fold cross validation

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.
- Then, split the data into k equal sets, each of N/k examples. Each of these sets is called a "fold."
- $k-1$ of the folds are datasets that are used to train the model parameters.
- The remaining fold is a testing 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



k-fold cross validation from housing example

Routines for training models and testing data.

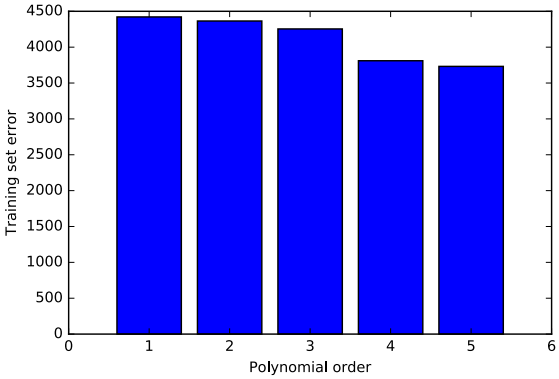
```
def train_models(x, y):
    # Fit a polynomial model up to N=5
    N = 5
    thetas = []
    for i in np.arange(N):
        xhat = np.vstack((x, np.ones_like(x))) if i == 0 else np.vstack((x**(i+1), xhat))
        thetas.append(np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y)))

    return thetas

def test_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
```

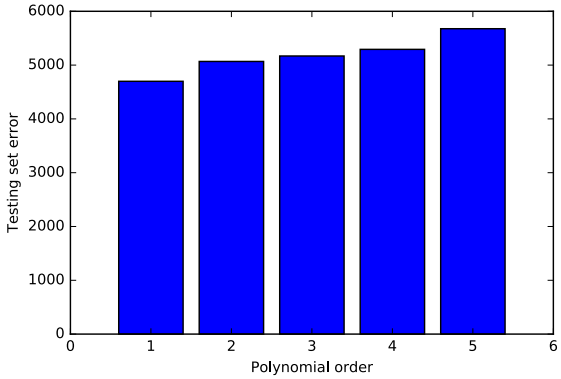
k-fold cross validation from housing example

```
# (x,y) are the training data pairs from the supervised housing rent example
# num_train = len(x) is the number of examples
num_folds = 5
cv_idx = np.arange(num_train)
np.random.shuffle(cv_idx)
fold_size = num_train // 5
train_error, test_error = (np.zeros(5), np.zeros(5))
for i in np.arange(num_folds):
    test_idx = cv_idx[i*fold_size:(i+1)*fold_size]
    train_idx = np.concatenate((cv_idx[:i*fold_size], cv_idx[(i+1)*fold_size:]))
    x_test = x[test_idx]
    y_test = y[test_idx]
    x_train = x[train_idx]
    y_train = y[train_idx]
    thetas = train_models(x_train, y_train)
    train_error += test_models(x_train, y_train, thetas)
    test_error += test_models(x_test, y_test, thetas)

print train_error
print test_error
f = plt.figure()
ax = f.gca()
ax.bar(np.arange(5) + 1, train_error, width=0.8, align='center')
ax.set_xlabel('Polynomial order')
ax.set_ylabel('Training set error')
f.savefig('ml-basics_cv-train.pdf')
f = plt.figure()
ax = f.gca()
ax.bar(np.arange(5) + 1, test_error, width=0.8, align='center')
ax.set_xlabel('Polynomial order')
ax.set_ylabel('Testing set error')
f.savefig('ml-basics_cv-test.pdf')
```

```
[ 4421.38242236  4364.65797472  4254.01787959  3810.5847096   3731.95920537]
[ 4698.83840383  5067.8036503   5169.2079499   5292.40392825  5676.26492547]
```


k-fold cross validation from housing example





Other types of optimization

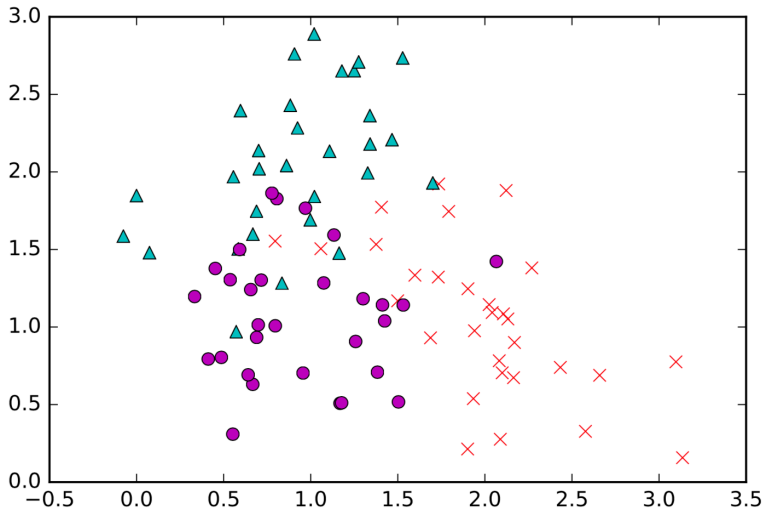
We've talked about examples where we want to *minimize* a mean-square error or distance metric.

Another metric that we may want to minimize 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.

Maximum-likelihood estimation

Example of ML estimation

Say we receive paired data $\{\mathbf{x}^{(i)}, y^{(i)}\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^2$ is a data point that belongs to one of three classes, $y^{(i)} \in \{1, 2, 3\}$. Classes $1, 2, 3$ denote the three possible classes (or labels) that a data point could belong to. The drawing below (with appropriate labels) represents this:



What is our goal in modeling?

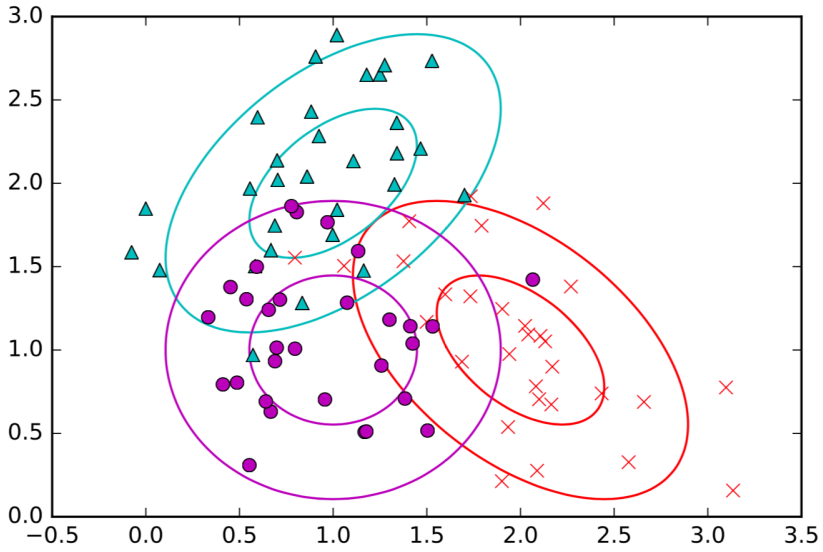
What is our goal in modeling?

Chain rule for probability

Chain rule for probability

Example of ML estimation (cont.)

We may want to model each cluster as being *generated* by a multivariate Gaussian distribution. Concretely, in this example, there are three Gaussian clusters, each one describing the distribution of points for that class.



Maximum-likelihood estimation

Example of ML estimation (cont.)

The model setup is as follows:

- Each data point $\mathbf{x}^{(i)}$ belongs to class $y^{(i)}$.
- Each class y_i is parametrized according to a distribution.

$$\begin{aligned}\mathbf{x}^{(i)}|y^{(i)}=1 &\sim \mathcal{N}(\mu_1,\Sigma_1)\\ \mathbf{x}^{(i)}|y^{(i)}=2 &\sim \mathcal{N}(\mu_2,\Sigma_2)\\ \mathbf{x}^{(i)}|y^{(i)}=3 &\sim \mathcal{N}(\mu_3,\Sigma_3)\end{aligned}$$

  Thus, the parameters we can choose to optimize our model are $\theta=\{\mu_1,\Sigma_1,\mu_2,\Sigma_2,\mu_3,\Sigma_3\}$.
- We'll assume all classes are equally probable a priori (so that maximum likelihood estimation and maximum a posteriori estimation are equivalent).
- Finally, we'll assume each data point is independent, so that we can easily write out probabilities, i.e.,

$$p\left(\{\mathbf{x}^{(i)},y^{(i)}\},\{\mathbf{x}^{(j)},y^{(j)}\}\right)=p\left(\mathbf{x}^{(i)},y^{(i)}\right)p\left(\mathbf{x}^{(j)},y^{(j)}\right)$$

Example of ML estimation (cont.)

We'd like to maximize the likelihood of having seen the dataset. This can be written as

$$\mathcal{L} = p\left(\{\mathbf{x}^{(1)}, y^{(1)}\}, \{\mathbf{x}^{(2)}, y^{(2)}\}, \ldots, \{\mathbf{x}^{(N)}, y^{(N)}\}\right)$$

Maximum-likelihood estimation

Example of ML estimation (cont.)

Now, to simplify the probabilities, we evaluate $\log p(\mathbf{x}_i|y_i)$. We first observe:

$$\log p(\mathbf{x}^{(i)}|y^{(i)}=j) \quad = \quad \log \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma_j|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j)\right)$$

Maximum-likelihood estimation

Maximum-likelihood estimation

$$\frac{\partial \log \mathcal{L}}{\partial \mu_j} = \frac{\partial}{\partial \mu_j} \left[\sum_{i:y^{(i)}=j} -\frac{1}{2}(\mathbf{x}^{(i)} - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}^{(i)} - \mu_j) \right]$$

Maximum-likelihood estimation

Examle of ML estimation (cont.)

We next take the derivative with respect to Σ, using the following facts (for more information, see "Tools" lecture notes on derivatives w.r.t. matrices):

$$\begin{aligned}\frac{\partial}{\partial \Sigma}\mathrm{tr}(\Sigma^{-1}\mathbf{A}) &= -\Sigma^{-T}\mathbf{A}^T\Sigma^{-T} \\ \frac{\partial}{\partial \Sigma}\log|\Sigma| &= \Sigma^{-T}\end{aligned}$$

Now, differentiating:

$$\frac{\partial \log \mathcal{L}}{\partial \Sigma_j} = \frac{\partial}{\partial \Sigma_j}\left[\sum_{i:y^{(i)}=j} -\frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j)\right]$$

Maximum-likelihood estimation

Example of ML estimation (cont.)

Solving for the optimal parameters μ_j and Σ_j for all classes, we arrive at the following solution:

```
# x and y are the data and their corresponding labels
# x is 2 x num_examples of the data points
# y is 1 x num_examples of the data labels

class_idxs = [np.where(y == j)[0] for j in np.unique(y)]
means = []
covs = []

for this_class in class_idxs:
    means.append(np.mean(x[:, this_class], axis=1))
    covs.append(np.cov(x[:, this_class]))
```


Maximum-likelihood estimation

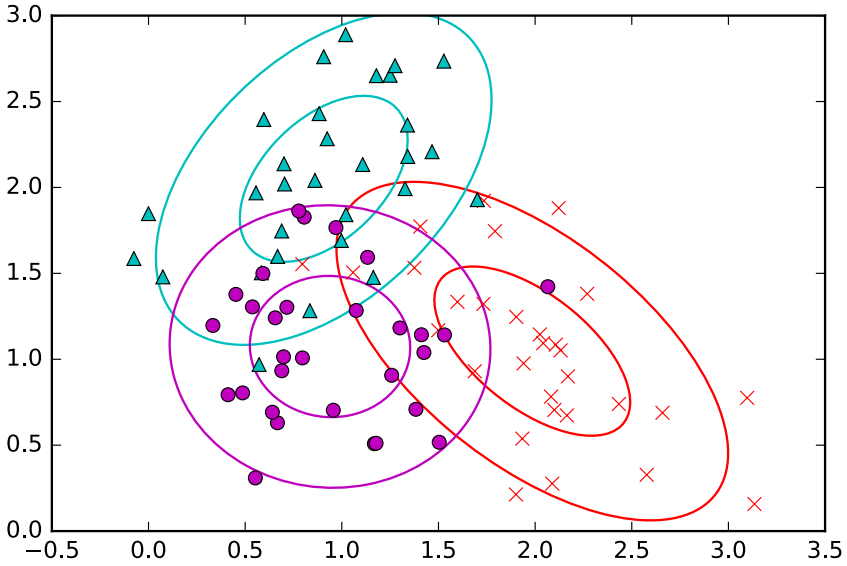
Example of ML estimation (cont.)

Solving for the optimal parameters μ_j and Σ_j for all classes, we arrive at the following solution:

```
f = plt.figure()
ax = f.gca()
ax.plot(data1[0,:], data1[1,:], marker='x', linestyle='', color=class_colors[0])
ax.plot(data2[0,:], data2[1,:], marker='^', linestyle='', color=class_colors[1])
ax.plot(data3[0,:], data3[1,:], marker='o', linestyle='', color=class_colors[2])


for i in np.arange(len(class_idxs)):
    vals, vecs = eigsorted(covs[i])
    theta = np.degrees(np.arctan2(*vecs[:,0][::-1]))
    for nstd in np.arange(3):
        w, h = 2 * nstd * np.sqrt(vals)
        e = Ellipse(xy=(means[i][0], means[i][1]),
                    width=w, height=h,
                    angle=theta, color=class_colors[i])
        e.set_facecolor('none')
        ax.add_artist(e)

f.savefig('ml-basics_maxlik-data_fit-ellipsoids.pdf')
```

Example of ML estimation (cont.)

Solving for the optimal parameters μ_j and Σ_j for all classes, we arrive at the following solution:



Maximum-likelihood estimation

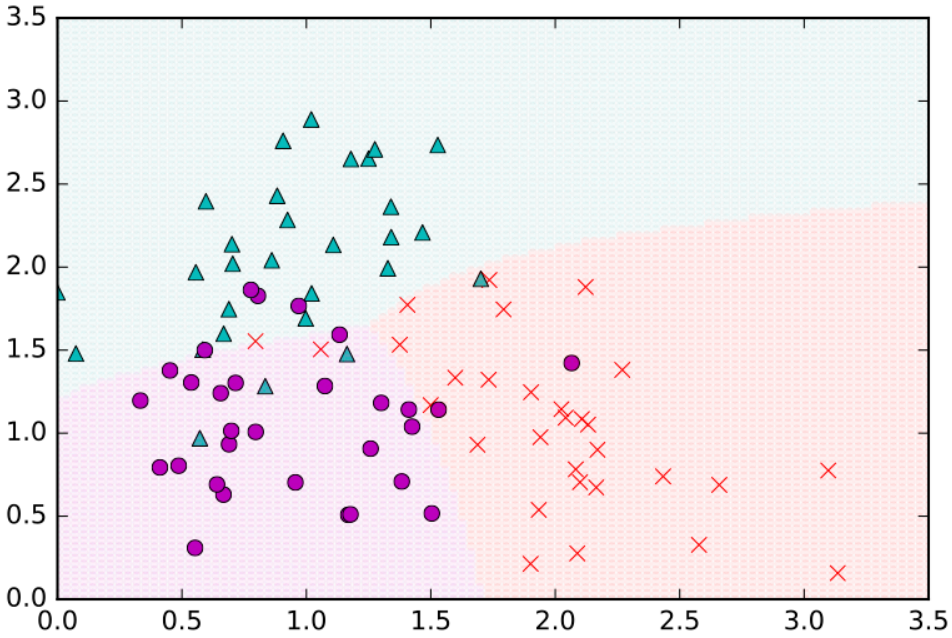
Example of ML estimation (cont.)

Imagine now a new point $\mathbf{x}$ comes in that we'd like to classify as being in one of three classes. To do so, we'd like to calculate: $\Pr(y = j|\mathbf{x})$ and pick the j that maximizes this probability. We'll denote this probability $p(j|\mathbf{x})$. We'll also assume, as earlier, that the classes are equally probable, i.e., $\Pr(y = j)$ is the same for all j.

Maximum-likelihood estimation

```
f = plt.figure()
ax = f.gca()
ax.plot(data1[0,:], data1[1,:], marker='x', linestyle='', color=class_colors[0])
ax.plot(data2[0,:], data2[1,:], marker='^', linestyle='', color=class_colors[1])
ax.plot(data3[0,:], data3[1,:], marker='o', linestyle='', color=class_colors[2])

for (i,xx) in enumerate(np.linspace(0, 3.5, 100)):
    for yy in np.linspace(0, 3.5, 100):
        xn = np.array((xx,yy))
        pmax = -np.Inf
        decoded_class = np.nan
        for c in np.arange(len(class_idxs)):
            class_prob = -np.log(np.linalg.det(covs[c])) - (xn-means[c]).T.dot(np.linalg.inv(covs[c])).dot(xn-means[c])
            if class_prob > pmax:
                pmax = class_prob
                decoded_class = c
        ax.plot(xx, yy, marker='.', color=class_colors[decoded_class], alpha=0.05)
ax.set_xlim(0,3.5)
f.savefig('ml-basics_maxlik-data_fit-ellipsoids_colorized.pdf')
```


Maximum-likelihood estimation



Even more cost functions

There are even more cost functions that we could use. We'll encounter some others in this class. Others include:

- MAP estimation.
- KL divergence.
- Maximize an approximation of the distribution.

In machine learning, it is important to arrive at an appropriate model and cost function. After that, it's important to know how to optimize it. In our examples here, the models were simple enough that we could differentiate and set the derivative equal to zero. In future lectures, we'll discuss more general ways to learn parameters of models when they are not so simple.