Basics of machine learning

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

Introduction

Machine learning involves using statistics to estimate (or learn) functions, some of which may be fairly complex. Some examples include:

- 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.

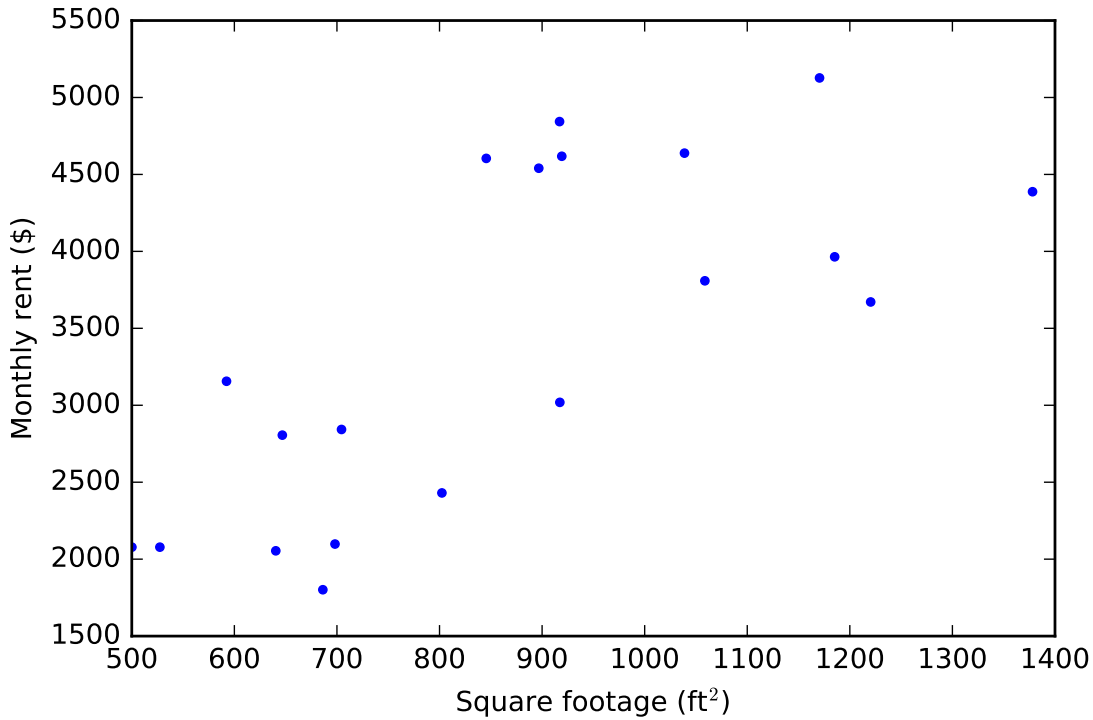
A setup for supervised learning

Consider a general supervised learning problem, where the goal is to predict targets $\mathbf{y}$ from inputs $\mathbf{x}$.

- To do this, we first define a *model*, f, parametrized by some set of variables θ.
- We wish to choose the settings of the variables in θ for which f "best predicts" $\mathbf{y}$ from $\mathbf{x}$, via $\mathbf{y} = f(\mathbf{x})$.
- The way we define "best predicts" is according to an *objective function*. The objective function quantifies the goals of the machine learning problem. In a classification problem, the objective may be to minimize the misclassification rate. In a regression problem, the objective may be to minimize the prediction squared loss.
- The parameters, θ, will then be found to optimize the objective function. One way this can be done is to find extrema of the objective function via differentiation.

Regression example (1D)

Let's say that we are given data point pairs (x, y). Our training set is composed of 100 of these data points. The data come in the plot below, and may represent, e.g., examples from the housing market of monthly rent as a function of square footage. (Note: this data is synthetic.) We let the ith data pair be denoted $(x^{(i)}, y^{(i)})$:



Regression example (1D, cont.)

Based off of some analysis of the data, or on prior knowledge, we decide that we want to fit a simple model:

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

where $\theta = (a, b)$ and $\hat{\mathbf{x}} = (x, 1)$. θ represents the parameters of the model, and we want to use our training examples to find the best values of θ (equivalently a and b) according to some cost function:

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

We wish to choose θ so that this cost function, $\mathcal{L}(\theta)$ is *smallest*. Hence, we wish to minimize $\mathcal{L}(\theta)$ with respect to θ.

Regression example (1D, cont.)

One way we know how to minimize a function with respect to a variable is to take its derivative and set it equal to zero. Note: this example is simple enough that we can use this approach. (Things won't always be so simple.)

Thus our strategy to find the best θ is to:

- Calculate
$$\frac{d\mathcal{L}}{d\theta}$$
- Solve for θ such that
$$\frac{\partial \mathcal{L}}{\partial \theta} = 0$$

Solving the optimization problem

(If unfamilliar with vector and matrix derivatives, please see the "Tools" notes on how to take derivatives with respect to vectors and matrices.)

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \theta} &= \sum_{i=1}^{N}(y^{(i)} - \theta^T \hat{\mathbf{x}}^{(i)})\hat{\mathbf{x}}^{(i)} \\ &\triangleq \mathbf{0}\end{aligned}$$

where $\mathbf{0}$ is a vector of zeros. This results in an undertermined systems of equations,

$$Y = \mathbf{X}\theta$$

for

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} (\hat{\mathbf{x}}^{(1)})^T \\ (\hat{\mathbf{x}}^{(2)})^T \\ \vdots \\ (\hat{\mathbf{x}}^{(N)})^T \end{bmatrix}$$

The solution is known as least-squares, i.e.,

$$\begin{aligned}\theta &= \mathbf{X}^{\dagger} Y \\ &= \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T Y\end{aligned}$$

Solving the optimization problem (aside)

In taking the vector derivatives, we used the *chain rule*. We'll discuss the chain rule more rigorously when we get to backpropagation, but we at least for now haven't formally derived a chain rule for gradients. Thus, one other way to solve this optimization problem would have been to a bit more algebra.

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

Solving the optimization problem (aside)

Now taking derivatives element by element, we have:

$$\frac{\partial \mathcal{L}}{\partial \theta} = -\mathbf{X}^T Y + \mathbf{X}^T \mathbf{X}\theta$$

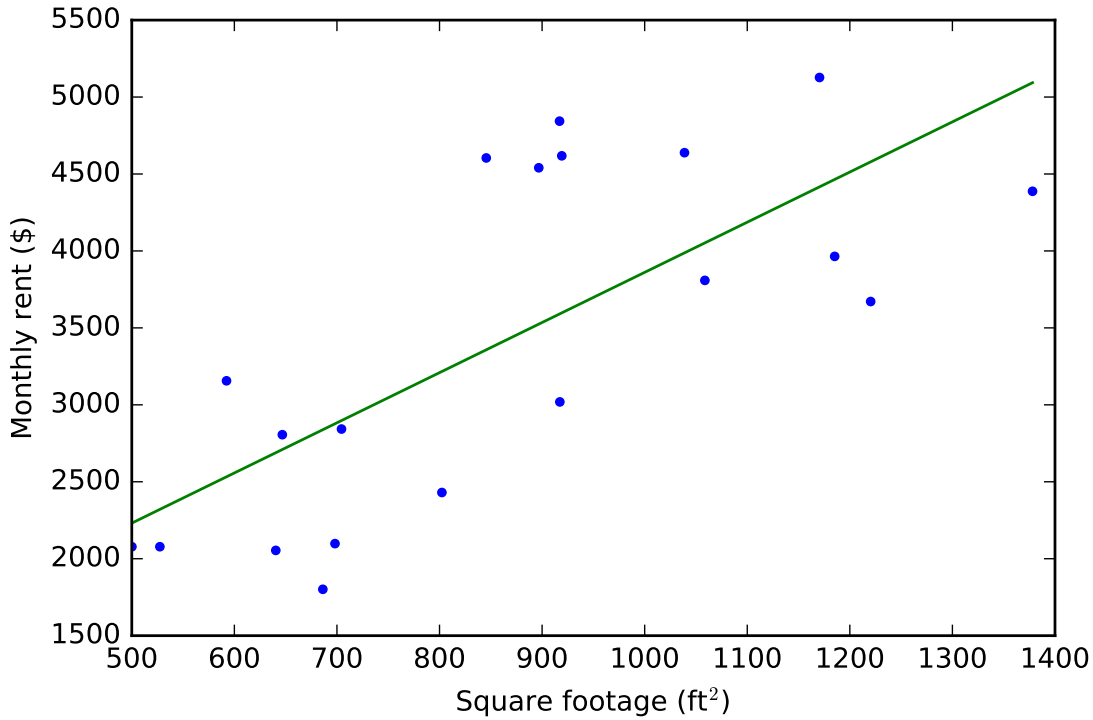
and equating to zero, we arrive at the same answer:

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

This solution is called the least-squares solution.

Regression example (solution)

Hence, the best linear fit of the data can be found via the least-squares solution:



This solution gets the general trend, but couldn't we do better at fitting a line through the blue points?

Generalizing to higher degree polynomials

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

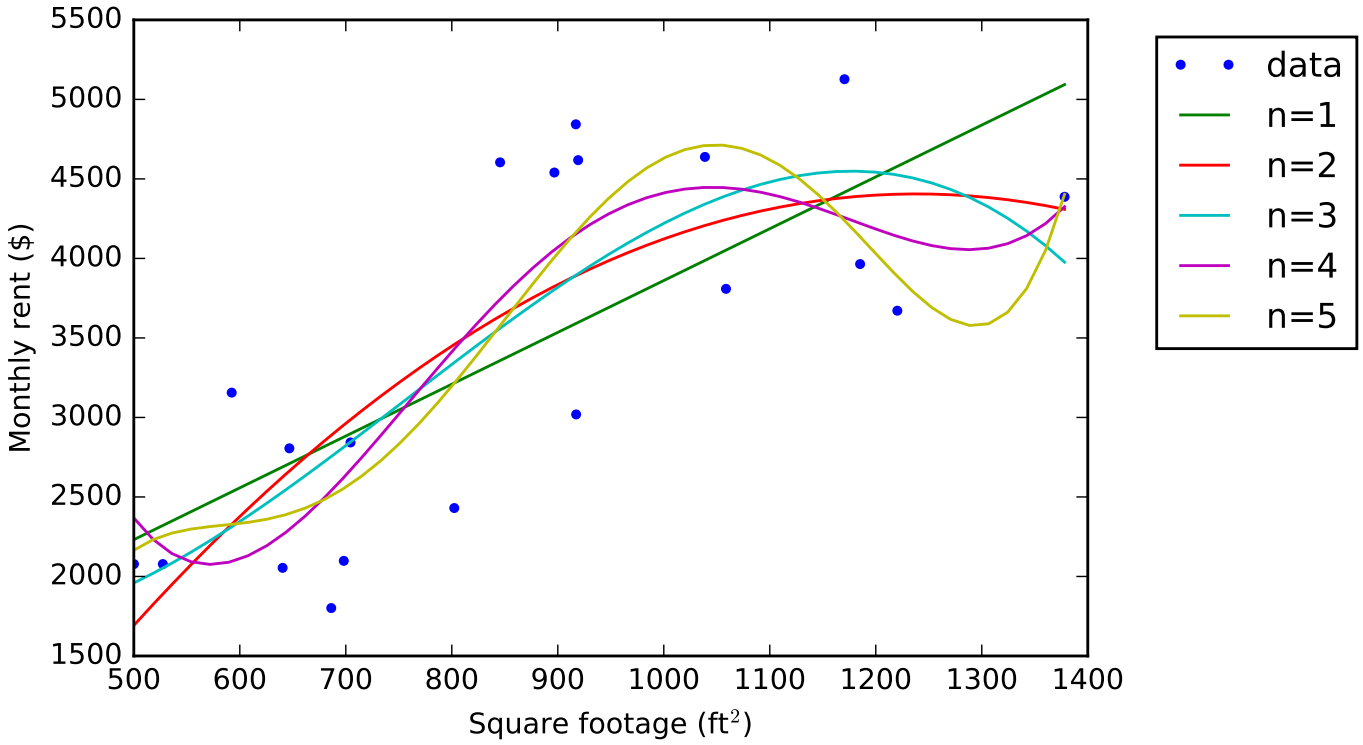
$$\begin{aligned} y &= b + a_1 x_1 + a_2 x^2 + \cdots + a_n x^n \\ &= \theta^T \hat{\mathbf{x}} \end{aligned}$$

for

$$\theta = \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_2 \\ a_1 \\ b \end{bmatrix} \quad \text{and } \hat{\mathbf{x}} = \begin{bmatrix} x^n \\ x^{n-1} \\ \vdots \\ x^2 \\ x \\ 1 \end{bmatrix}$$

Regression example (higher order)

A higher order polynomial will always fit the data better (according to the desired cost function). Why?

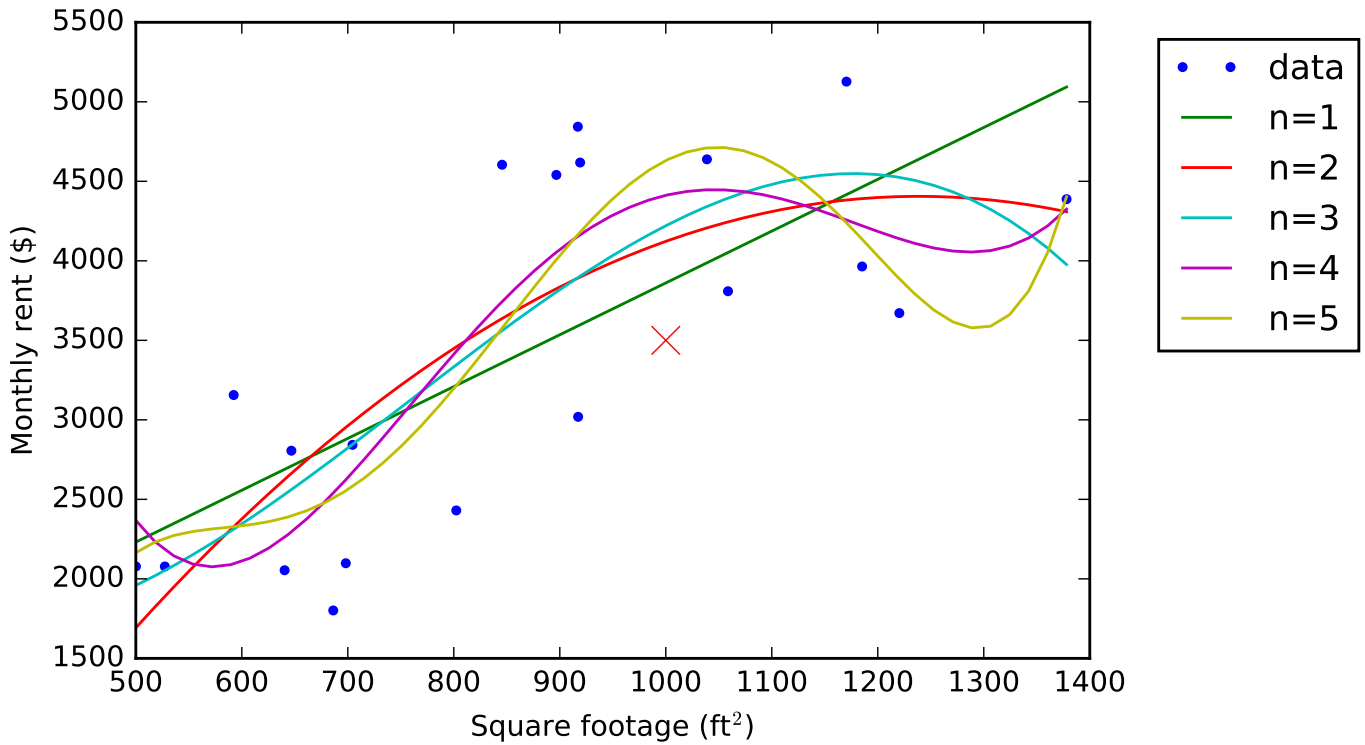


Model complexity (capacity)

This example begs the question: doesn't the model play a very large role?

- It appears straightforward to arbitrarily reduce the training error in the regression example.
- Namely, we can make the model an arbitrarily complex polynomial.

However, the model may now not generalize as well; see new data point 'x' which is not unreasonable, and yet the linear model performs best on it.



Overfitting

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

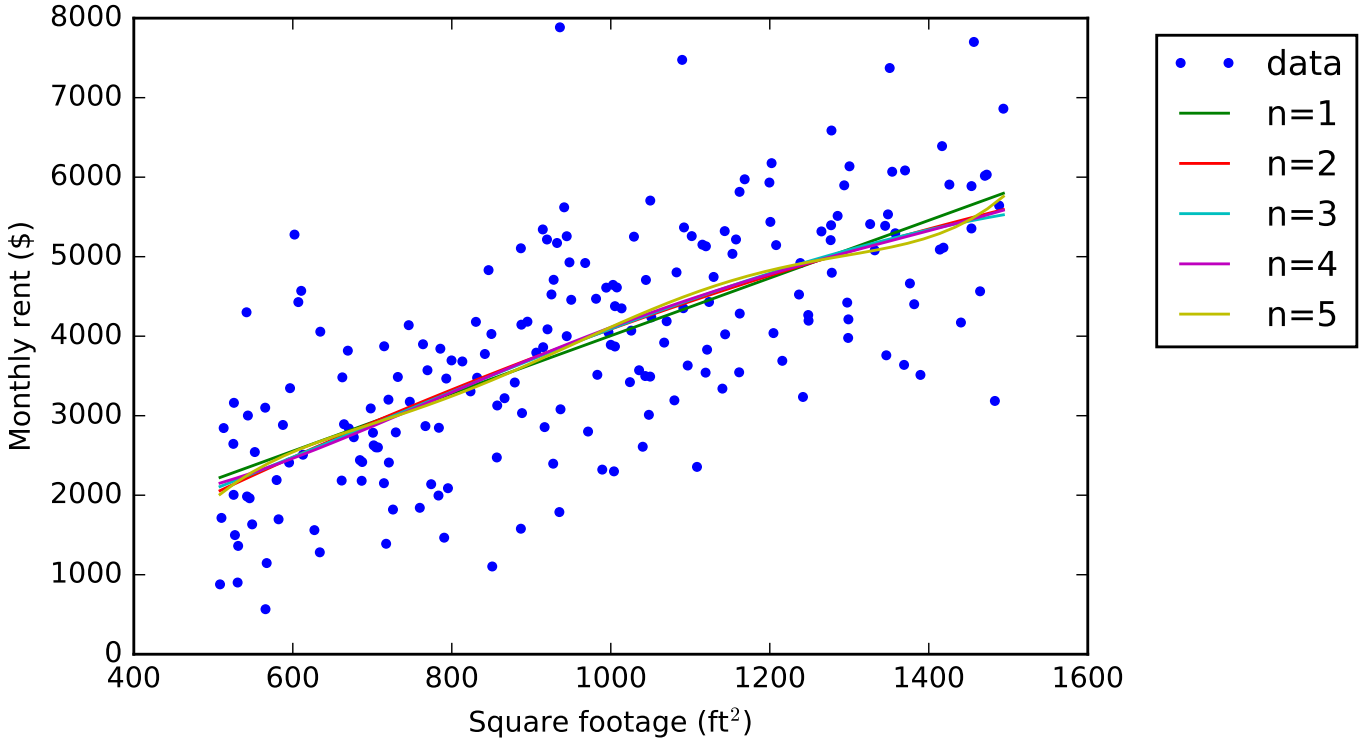
- **Training data** is data that is used to learn the parameters of your model.
- **Testing data** is data that is excluded in training and used to score your model.

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*.

Larger datasets ameliorate overfitting

While overfitting may arise due to an overly complex model, it can be helped by incorporating more training data.

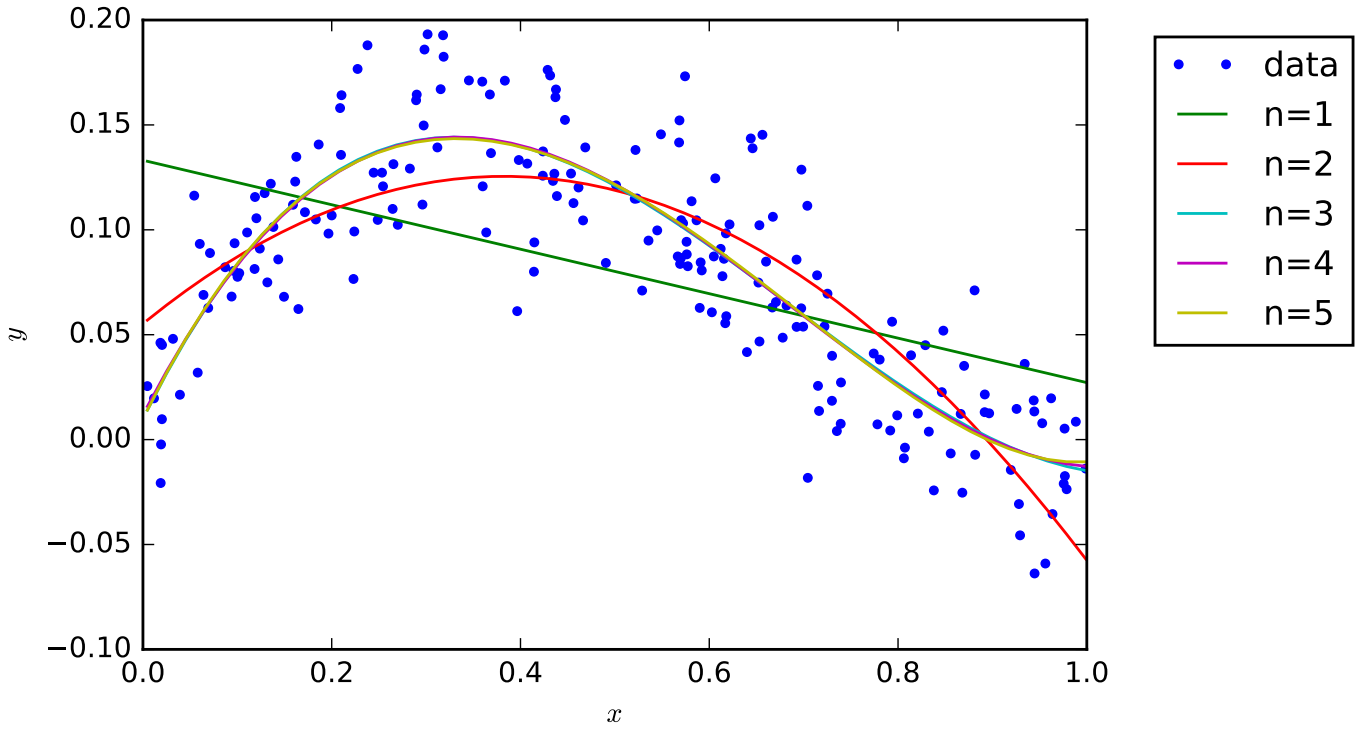


This suggests that when *a lot* of data is available, it may be appropriate to use more complex models. (Another technique we will discuss later, regularization, also helps with overfitting.) This is a shadow of things to come (i.e., neural networks which have large complexity).

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.

Below: data generated from a cubic model, fit with different order polynomial models.



Estimators

The notions of underfitting and overfitting are closely related to two other concepts: bias and variance. The bias and variance quantify important statistics about an *estimator*.

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 $\text{bias}(\hat{\theta}_m) = 0$.
- An estimator is called *asymptotically unbiased* if $\lim_{m\to\infty} \mathbb{E}(\hat{\theta}_m) = \theta$, i.e., $\text{bias}(\hat{\theta}_m) \to 0$.

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:

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

Then, the bias of this estimator is:

$$\begin{aligned} \text{bias}(\hat{\theta}_m) &= \mathbb{E}\left(\frac{1}{m} \sum_{i=1}^{m} x^{(i)} \right) - \theta \\ &= \frac{1}{m} \sum_{i=1}^{m} \mathbb{E} x^{(i)} - \theta \\ &= \frac{1}{m} \sum_{i=1}^{m} \theta - \theta \\ &= 0 \end{aligned}$$

where we used the fact that

$$\mathbb{E} x^{(i)} = \sum_{x^{(i)} \in \{0,1\}} x^{(i)} p(x^{(i)}) = \sum_{x^{(i)} \in \{0,1\}} x^{(i)} \theta^{x^{(i)}} (1-\theta)^{(1-x^{(i)})}$$

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 $\mathrm{var}(\hat{\theta})$.

Some notes:

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

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:

$$\begin{aligned}
\text{var}(\hat{\theta}_m) &= \text{var}\left(\frac{1}{m} \sum_{i=1}^{m} x^{(i)} \right) \\
&= \frac{1}{m^2} \sum_{i=1}^{m} \text{var}\left(x^{(i)} \right) \\
&= \frac{1}{m^2} \sum_{i=1}^{m} \theta(1-\theta) \\
&= \frac{\theta(1-\theta)}{m}
\end{aligned}$$

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.

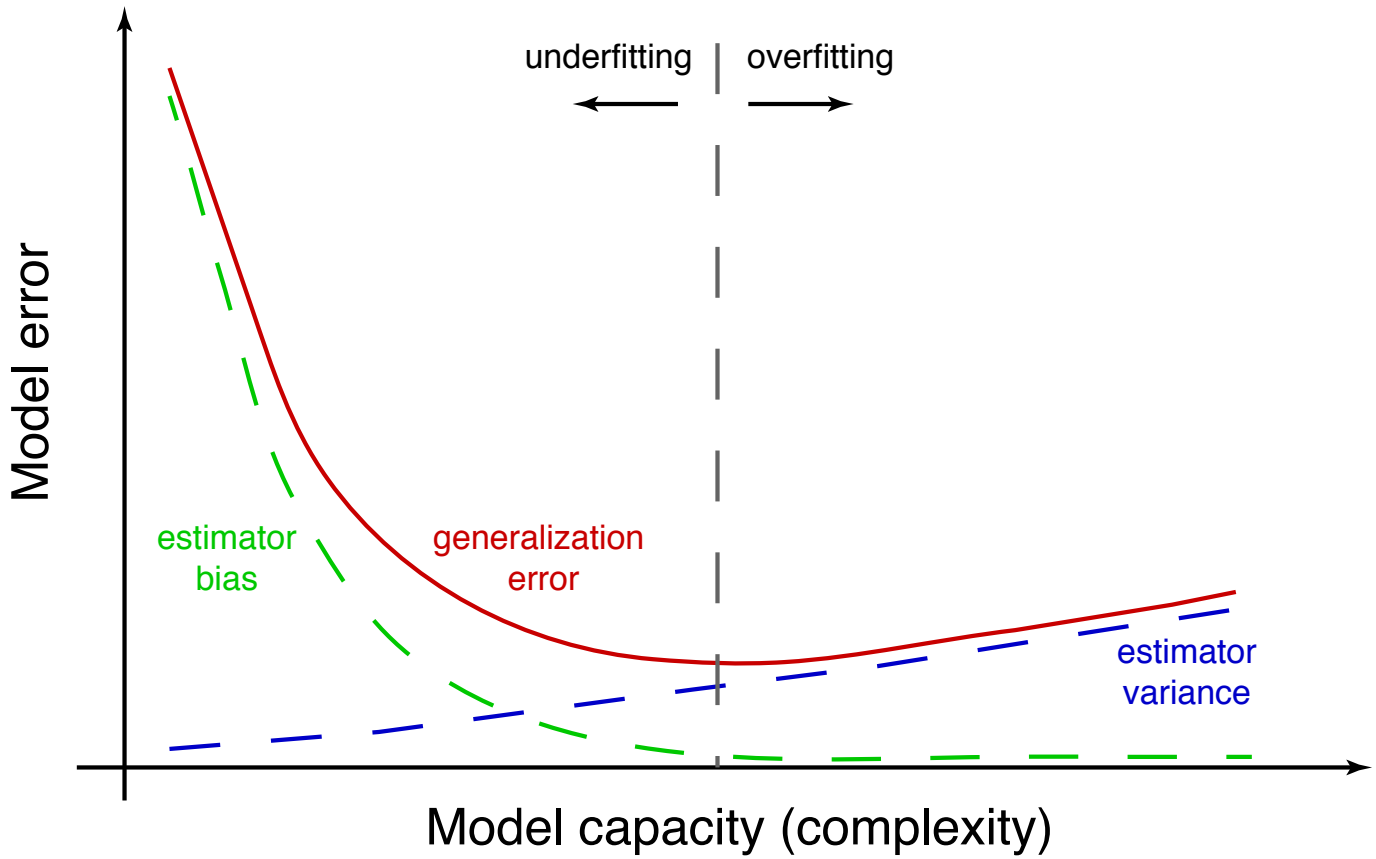
$$\begin{aligned}
\text{MSE} &= \mathbb{E}(\hat{\theta}_m - \theta)^2 \\
&= \mathbb{E}(\hat{\theta}_m - \mathbb{E}\hat{\theta}_m + \mathbb{E}\hat{\theta}_m - \theta)^2 \\
&= \mathbb{E}(\hat{\theta}_m - \mathbb{E}\hat{\theta}_m)^2 + \mathbb{E}\left[2(\hat{\theta}_m - \mathbb{E}\hat{\theta}_m)(\mathbb{E}\hat{\theta}_m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

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

Model capacity, bias and variance

When we measure the performance of an estimator via MSE ...

- ... a model that underfits the data has high bias.
- ... a model that overfits the data has high variance.



Choosing a model

Thus, we arrive at a problem. How can we select the best model if these can be made arbitrarily complex?

- In the scenario where no data was set aside and we must make an evaluation on the training data only, this is the problem of model selection. There are criterion which penalize the model complexity. For example ...
  - Akaike information criterion (AIC)
  - Bayes information criterion (BIC)
  - Deviance informaiton criterion (DIC)
  - We won't cover these in class.
- More typically, we leave aside data that was not used in training, or generate new data. We then *test* our model on this non-training set data, and evaluate the model's performance.

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.

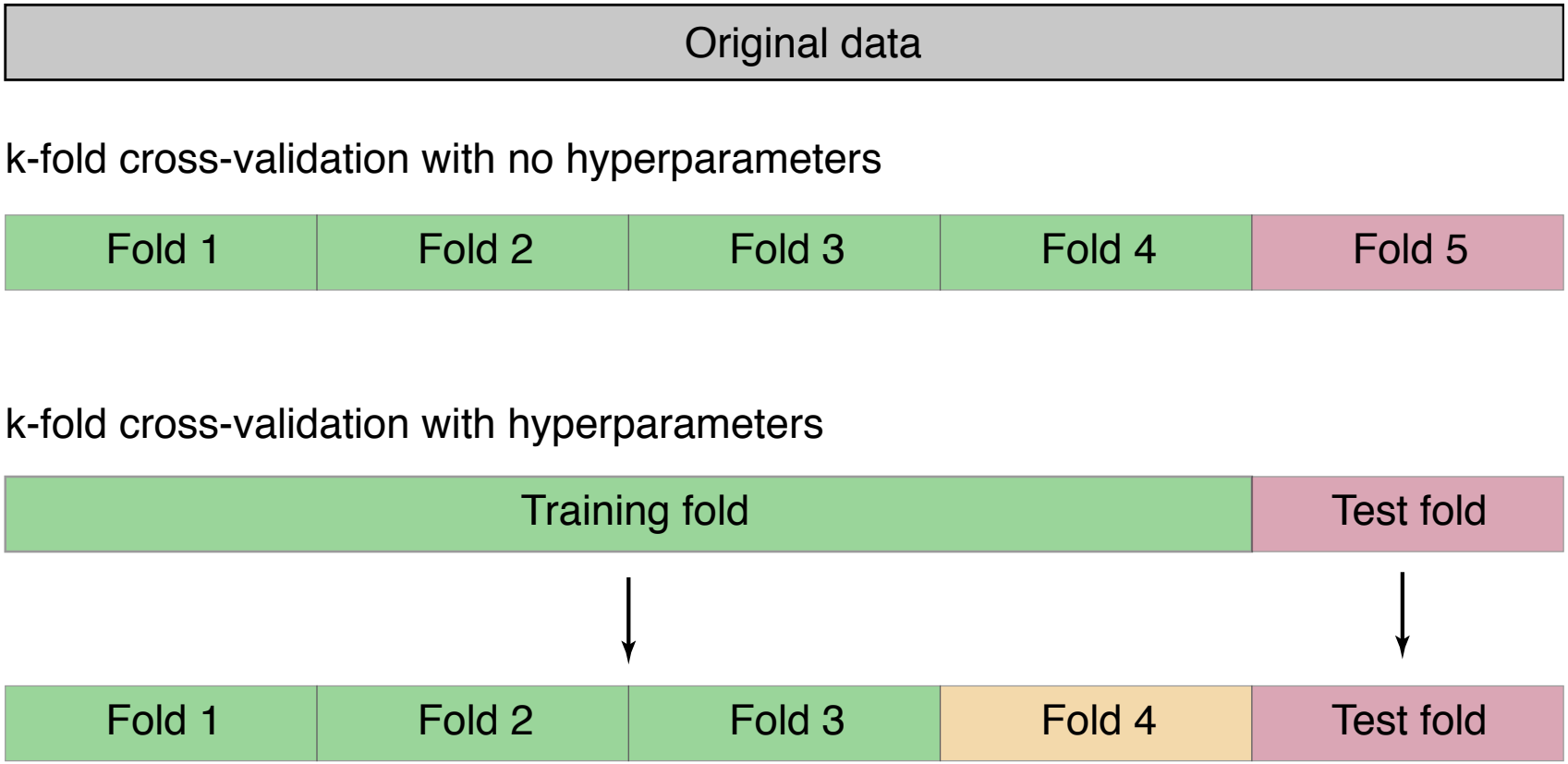
k-fold cross validation

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.

k-fold cross validation

The following picture is appropriate for k-fold cross-validation. Green denotes training data, red denotes testing data, and yellow denotes validation data.



In practice, k-fold cross-validation may be expensive and hence not appropriate.

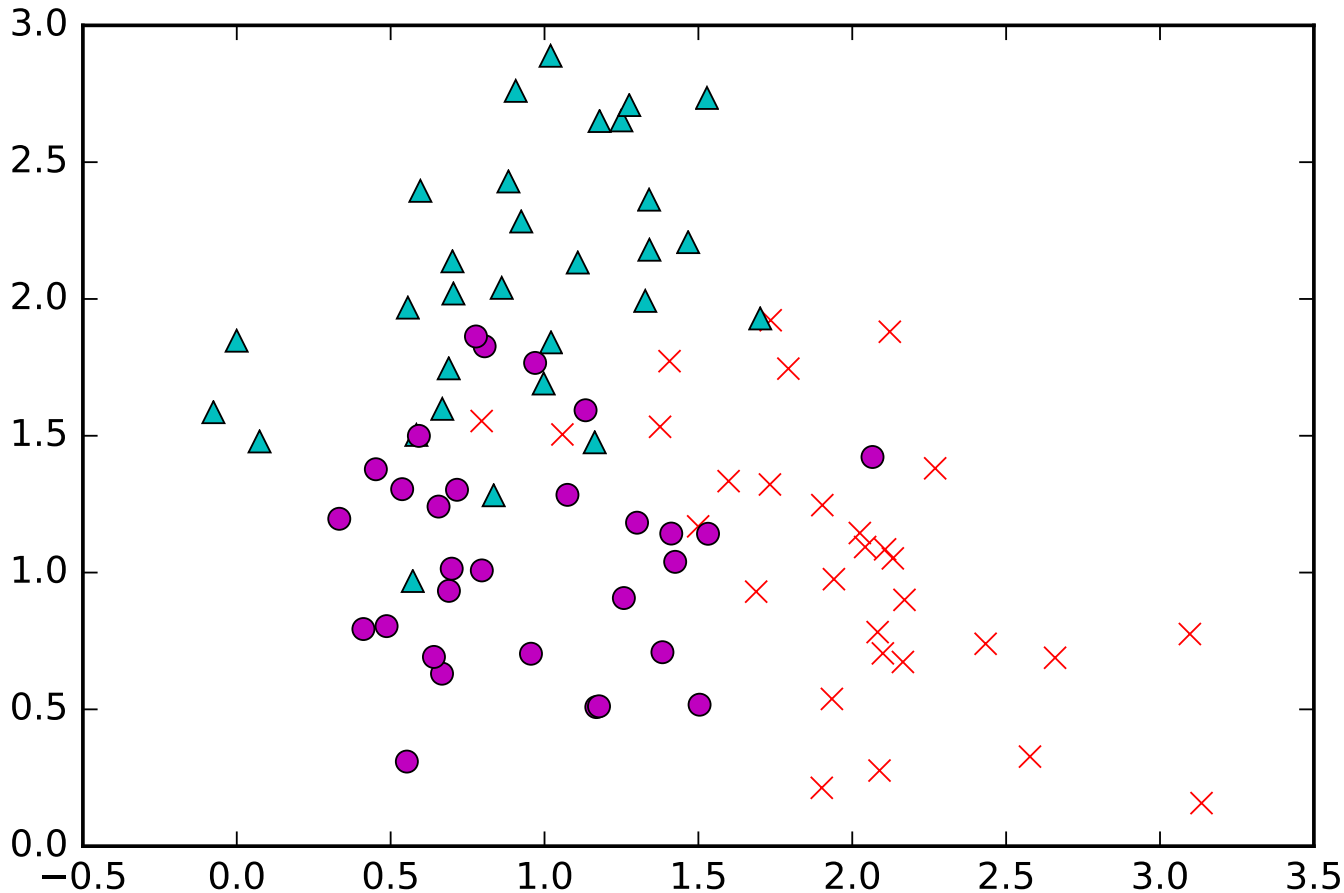
Other types of optimization

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.

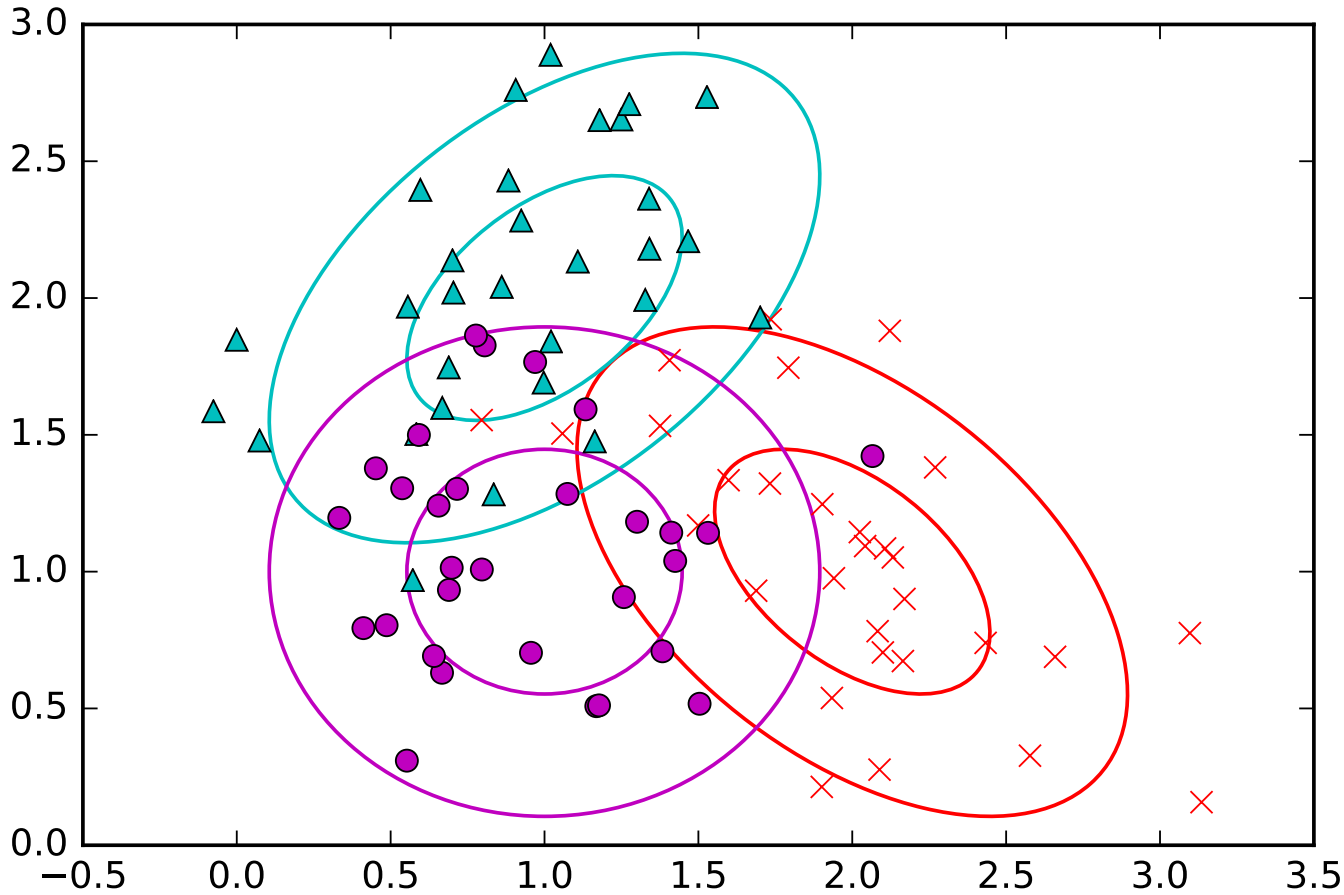
Example of ML estimation

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:



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.



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

$$\begin{aligned}
\mathcal{L} &= p\left(\{\mathbf{x}^{(1)}, y^{(1)}\}, \{\mathbf{x}^{(2)}, y^{(2)}\}, \ldots, \{\mathbf{x}^{(N)}, y^{(N)}\}\right) \\
&= \prod_{i=1}^{N} p(\mathbf{x}^{(i)}, y^{(i)}) \\
&= \prod_{i=1}^{N} p(y^{(i)}) p(\mathbf{x}^{(i)} | y^{(i)})
\end{aligned}$$

Because $\log(\cdot)$ is a monotonic function, a common trick in writing objective functions with probabilities is to take the log of the probability, thus turning products into sums.

$$\begin{aligned}
\log \mathcal{L} &= \sum_{i=1}^{N} \left[\log p(y^{(i)}) + \log p(\mathbf{x}^{(i)} | y^{(i)})\right] \\
&= k_1 + \sum_{i=1}^{N} \log p(\mathbf{x}^{(i)} | y^{(i)})
\end{aligned}$$

Example of ML estimation (cont.)

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

$$\begin{aligned}
\log p(\mathbf{x}^{(i)}|y^{(i)}=j) &= \log \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma_j|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j)\right) \\
&= -\frac{n}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j) \\
&= k_2 - \frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j)
\end{aligned}$$

Hence, we arrive at:

$$\log \mathcal{L} = k + \sum_{i=1}^{N} -\frac{1}{2}\log|\Sigma_{y^{(i)}}| - \frac{1}{2}(\mathbf{x}^{(i)}-\mu_{y^{(i)}})^T\Sigma_{y^{(i)}}^{-1}(\mathbf{x}^{(i)}-\mu_{y^{(i)}})$$

where $k = k_1 + Nk_2$.

Examle of ML estimation (cont.)

Let's then find the optimal parameters by differentiating and setting to zero. First, we find the optimal μ_j. We note that only the last term in the log likelihood has μ_j terms in instances where $y^{(i)} = j$.

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

Setting the derivative to zero, we have that:

$$\mu_j = \frac{1}{N_j} \sum_{i:y^{(i)}=j} \mathbf{x}^{(i)}$$

where $N_j = \sum_{i:y^{(i)}=j} 1$, i.e., the number of training examples where $y^{(i)} = j$. Does this answer make sense?

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:

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

Example of ML estimation (cont.)

Setting the derivative equal to zero, we arrive at:

$$\sum_{i:y^{(i)}=j} \Sigma_j^{-1} = \sum_{i:y^{(i)}=j} \Sigma_j^{-1}(\mathbf{x}^{(i)} - \mu_j)(\mathbf{x}^{(i)} - \mu_j)^T \Sigma_j^{-1}$$

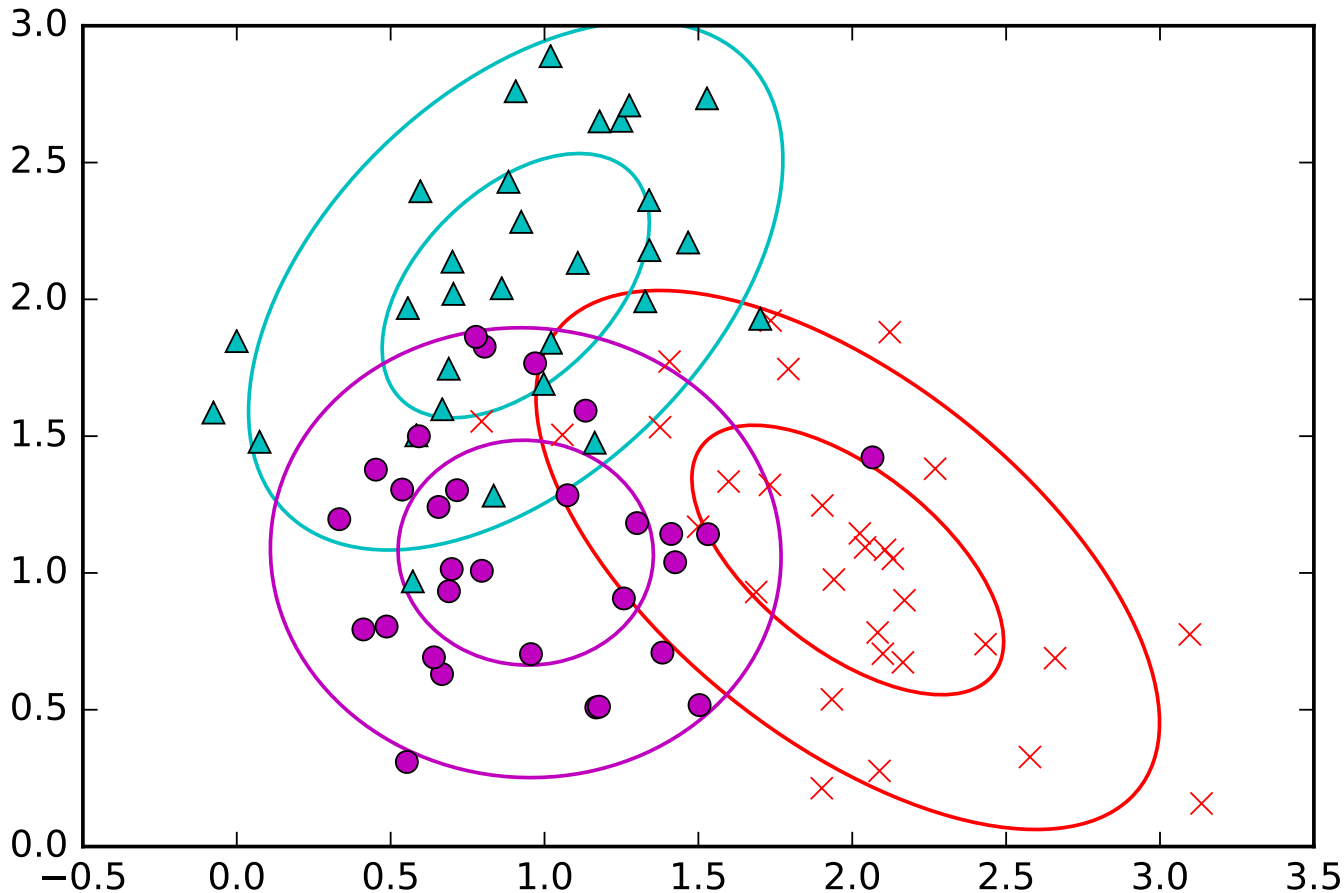
Pre and post-multiplying both sides by Σ_j, we then have:

$$\Sigma_j = \frac{1}{N_j} \sum_{i:y^{(i)}=j} (\mathbf{x}^{(i)} - \mu_j)(\mathbf{x}^{(i)} - \mu_j)^T$$

Does this answer make sense?

Example of ML estimation (cont.)

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



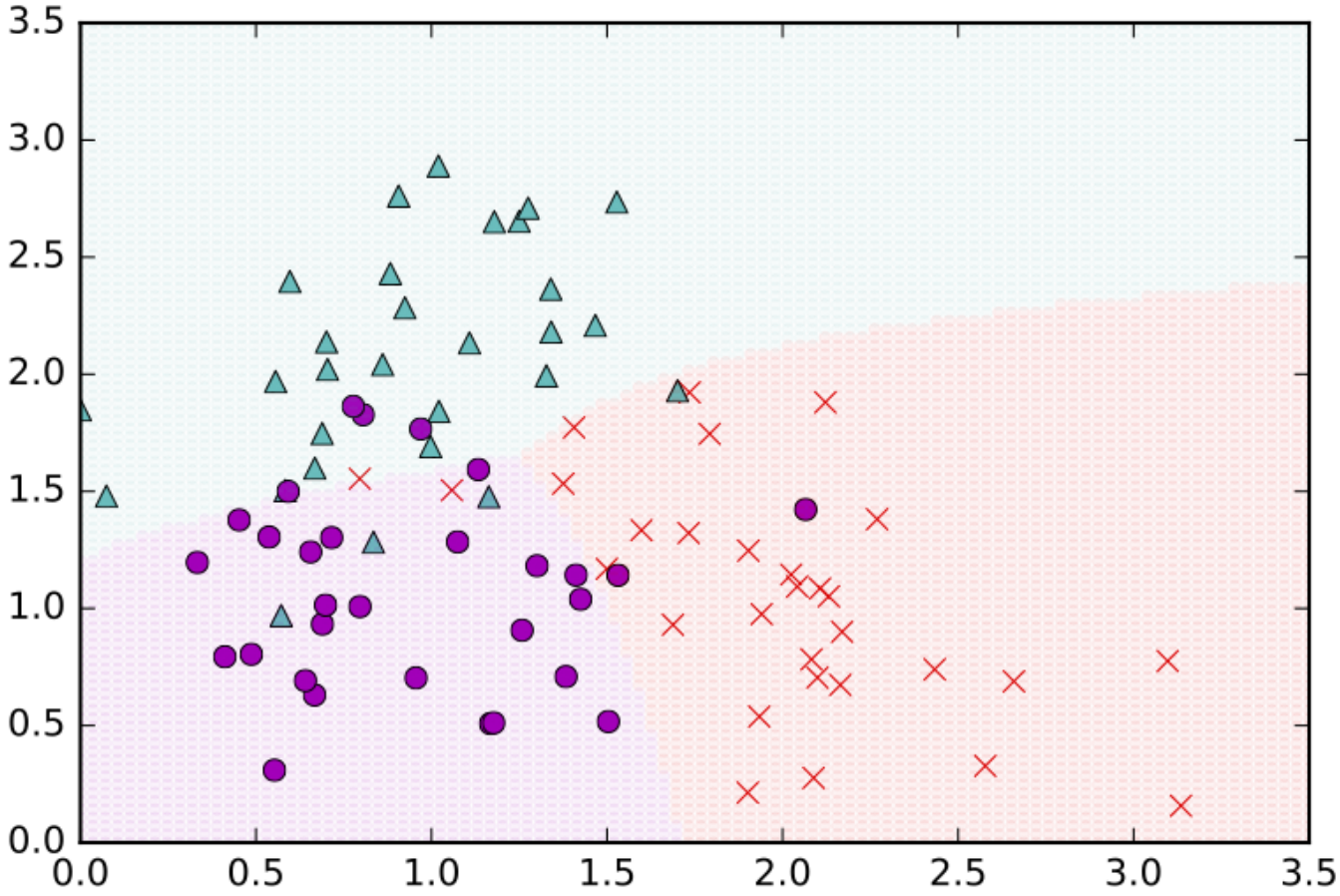
Example of ML estimation (cont.)

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.

$$\begin{aligned}
\arg\max_j p(j|\mathbf{x}) &= \arg\max_j \frac{p(\mathbf{x}|j)p(j)}{p(\mathbf{x})} \\
&= \arg\max_j p(\mathbf{x}|j)p(j) \\
&= \arg\max_j \log p(\mathbf{x}|j) \\
&= \arg\max_j \left[-\log|\Sigma_j| - (\mathbf{x} - \mu_j)^T \Sigma_j^{-1} (\mathbf{x} - \mu_j) \right]
\end{aligned}$$

Example of ML estimation (cont.)

This classification rule results in the following classifier:



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 cost functions were simple enough (i.e., quadratic in the parameters) 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.