JOHNS HOPKINS UNIVERSITY

EN.600.475 Machine Learning

General Linear Regression

Raman Arora
Lecture 7
February 17, 2017

- Polynomial regression
- Gradient descent

Slides credit: Greg Shakhnarovich

Polynomial regression

- Consider 1D for simplicity:

$$f(x;\mathbf{w}) \;=\; w_0 + w_1 x + w_2 x^2 + \ldots + w_m x^m.$$

- No longer linear in x – but still linear in $\mathbf{w}$!
- Define $\boldsymbol{\phi}(x) = [1, x, x^2, \ldots, x^m]^T$
- Then, $f(x;\mathbf{w}) = \mathbf{w} \cdot \boldsymbol{\phi}(x)$ and we are back to the familiar simple linear regression. The least squares solution:

$$\hat{\mathbf{w}} \;=\; \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}, \text{ where } \mathbf{X} \;=\; \begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^m \\ 1 & x_2 & x_2^2 & \ldots & x_2^m \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_N & x_N^2 & \ldots & x_N^m \end{bmatrix}$$

General additive regression models

- A general extension of the linear regression model:

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1\phi_1(\mathbf{x}) + w_2\phi_2(\mathbf{x}) + \ldots + w_m\phi_m(\mathbf{x}),$$

where $\phi_j(\mathbf{x}) : \mathcal{X} \to \mathbb{R}, j = 1, \ldots, m$ are the *basis functions*.

- This is still linear in $\mathbf{w}$,

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x})$$

even when $\boldsymbol{\phi}$ is non-linear in the inputs $\mathbf{x}$.

General linear regression

General additive regression models

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1\phi_1(\mathbf{x}) + w_2\phi_2(\mathbf{x}) + \ldots + w_m\phi_m(\mathbf{x}),$$

- Still the same ML estimation technique applies:

$$\hat{\mathbf{w}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

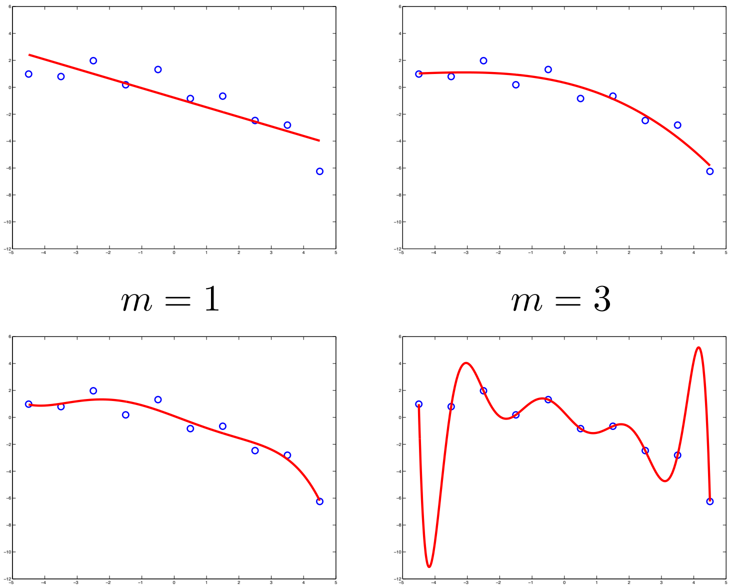
where $\mathbf{X}$ is the *design matrix*

$$\begin{bmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \ldots & \phi_m(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \ldots & \phi_m(\mathbf{x}_2) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \phi_2(\mathbf{x}_N) & \ldots & \phi_m(\mathbf{x}_N) \end{bmatrix}$$

(for convenience we will denote $\phi_0(\mathbf{x}) \equiv 1$)

Model complexity and overfitting

- Data drawn from 3rd order model:



$m = 1$ $m = 3$

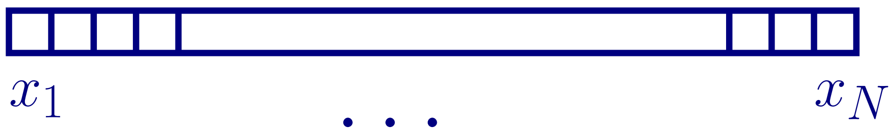
$m = 5$ $m = 10$

How to avoid overfitting

- The basic idea: if a model overfits (is *too sensitive* to data) it will be unstable. I.e. removal part of the data will change the fit significantly.
- We can *hold out* part of the data; this is *validation (val)* set, or *development (dev)* set.
- Fit the model to the rest, and then test on the heldout data.
- What are the problems of this approach?
  - If the heldout set too small, we are susceptible to chance.
  - If it's too large, we get overly pessimistic (training on too little data compared to what we could do).

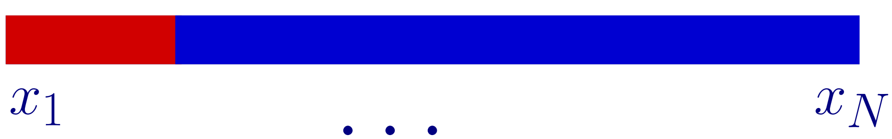
Cross-validation

- The improved holdout method: k-fold *cross-validation*
  - Partition data into k roughly equal parts;
  - Train on all but j-th part, test on j-th part



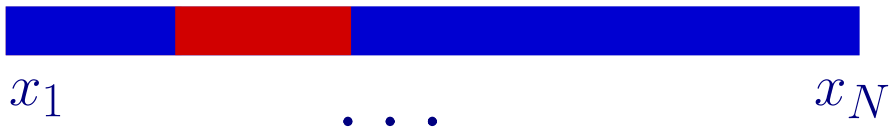
Cross-validation

- The improved holdout method: k-fold *cross-validation*
  - Partition data into k roughly equal parts;
  - Train on all but j-th part, test on j-th part



Cross-validation

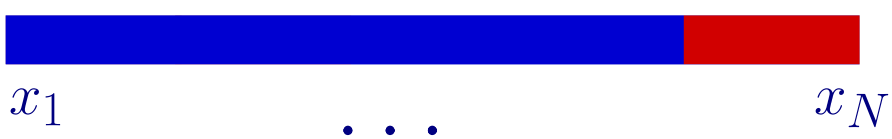
- The improved holdout method: k-fold *cross-validation*
  - Partition data into k roughly equal parts;
  - Train on all but j-th part, test on j-th part



Overfitting and complexity

Cross-validation

- The improved holdout method: k-fold *cross-validation*
  - Partition data into k roughly equal parts;
  - Train on all but j-th part, test on j-th part



Cross-validation

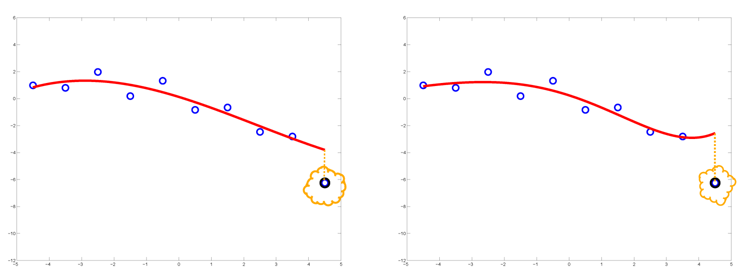
- The improved holdout method: k-fold *cross-validation*
  - Partition data into k roughly equal parts;
  - Train on all but j-th part, test on j-th part

- An extreme case: *leave-one-out* cross-validation

$$\hat{L}_{\text{cv}} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f(\mathbf{x}_i; \hat{\mathbf{w}}_{-i})\right)^2$$

where $\hat{\mathbf{w}}_{-i}$ is fit to all the data but the i-th example.

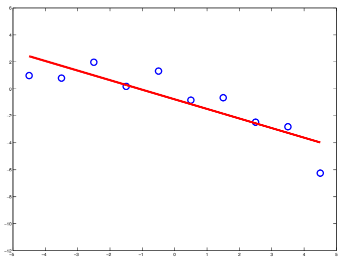
Cross-validation: example



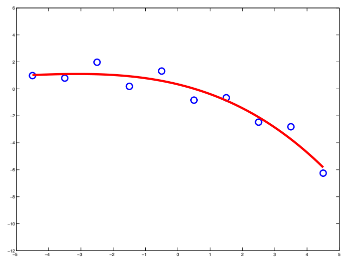
$m = 3$ $\qquad$ $m = 5$

- This is a very good estimate, although expensive to compute
  - Need to run N estimation problems each on $N - 1$ examples!
  - An important research area: devising tricks for efficiently computing cross-validation estimates (by taking advantage of overlap between folds).

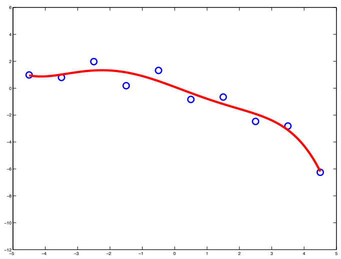
Cross-validation: example



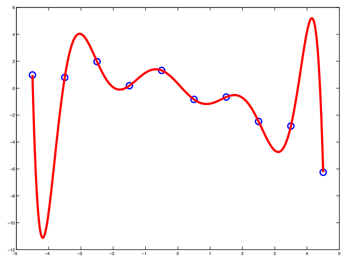
$m = 1 : L = 1.4, \hat{L}_{\text{CV}} = 2.6$



$m = 3 : L = 0.4, \hat{L}_{\text{CV}} = 1.3$



$m = 5 : L = 0.3, \hat{L}_{\text{CV}} = 2.7$



$m = 10 : L = 0, \hat{L}_{\text{CV}} = 4 \times 10^4$

Understanding overfitting

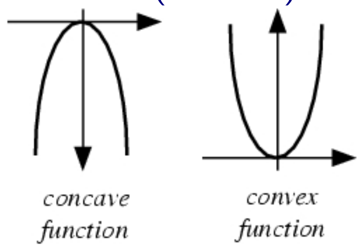
- Cross validation provides some means of dealing with overfitting
- What is the source of overfitting? Why do some models overfit more than others?
- We can try to get some insight by thinking about the estimation process for model parameters

Beyond closed form solution

- So far: solve (least squares) regression with a closed form solution

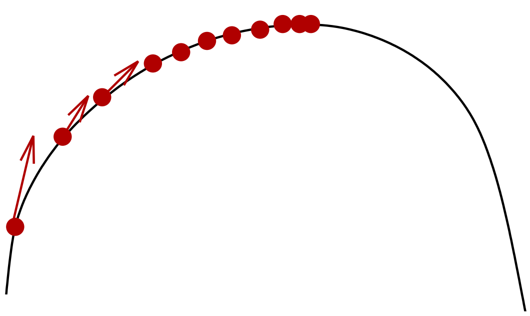
$$\mathbf{w}^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

- Sometimes we can not do this. E.g., the data matrix is too large to compute the pseudoinverse for
- If we move away from simple squared loss (e.g., in PS1: asymmetric loss) also lose the closed form solution
- Alternative: numerical optimization – gradient descent
- Consider (for now) convex or concave functions



Gradient ascent/descent

- The idea behind gradient ascent: "hill climbing" on the function surface.



- Start at a (random) location
- Make steps in the direction of maximal altitude increase.

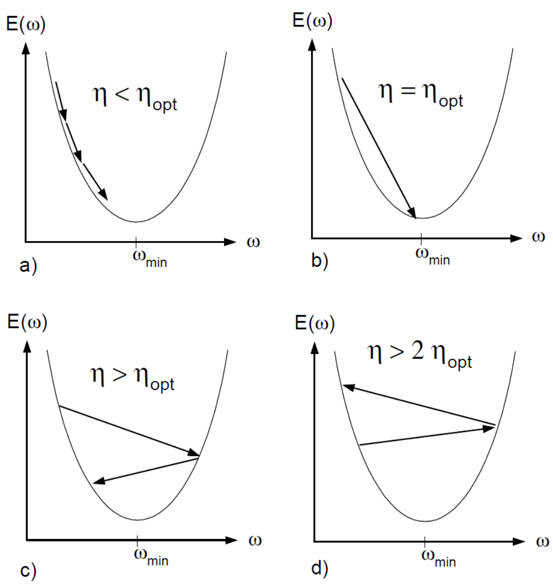
- An equivalent: gradient *descent* on the *convex* loss $-\log p\left(y \mid \mathbf{x}; \mathbf{w}\right)$

Gradient descent algorithm on $f(\mathbf{X}, \mathbf{y}; \mathbf{w})$

- Iteration counter $t = 0$
- Initialize $\mathbf{w}^{(t)}$ (to zero or a small random vector)
- for $t = 1, \ldots$:
  compute gradient
  $$\mathbf{g}^{(t)} = \nabla f\left(\mathbf{X}, \mathbf{y}; \mathbf{w}^{(t-1)}\right)$$
  update model
  $$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \eta \mathbf{g}^{(t)}$$
  check for convergence
- The *learning rate* η controls the step size

Gradient descent convergence

- Generally, for convex functions, gradient descent will converge
- Setting the learning rate η may be very important to ensure rapid convergence



From Lecun et al, 1996

Gradient descent convergence

- A lot of theory on convergence of gradient descent
- Usually relies on various properties of the objective function: strong convexity, smoothness, etc.
- In practice, need to monitor the objective, tweak learning rate, and consider stopping ("convergence") criteria
- Common criteria (often use a combination):
  - Maximum number of iterations (time budget)
  - Minimum required change in objective value (loss)
  - Minimum required change in model parameters ($\mathbf{w}$)
- If stopped because of max iterations: may not have converged
- Problematic criteria: monitor absolute (not relative) value of something like objective or parameters. Often hard to know what the "right" value for these is.