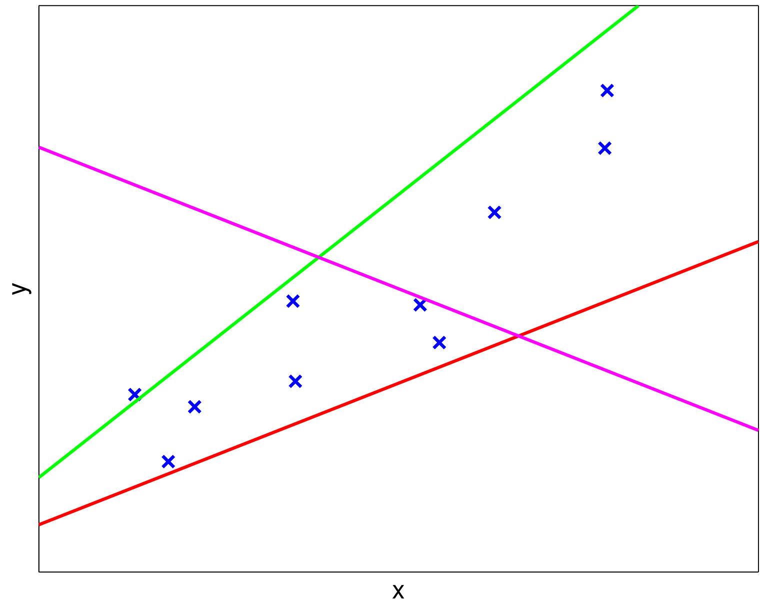
COMP 652: Machine Learning

Lecture 2

Today

- Continuing linear regression
  - Reminder of the problem
  - Finding the optimal weights
- Polynomial regression
  - Formulating and solving the problem
  - Overfitting / generalization error
  - Cross-validation
- The maximum likelihood principle (or heuristic)

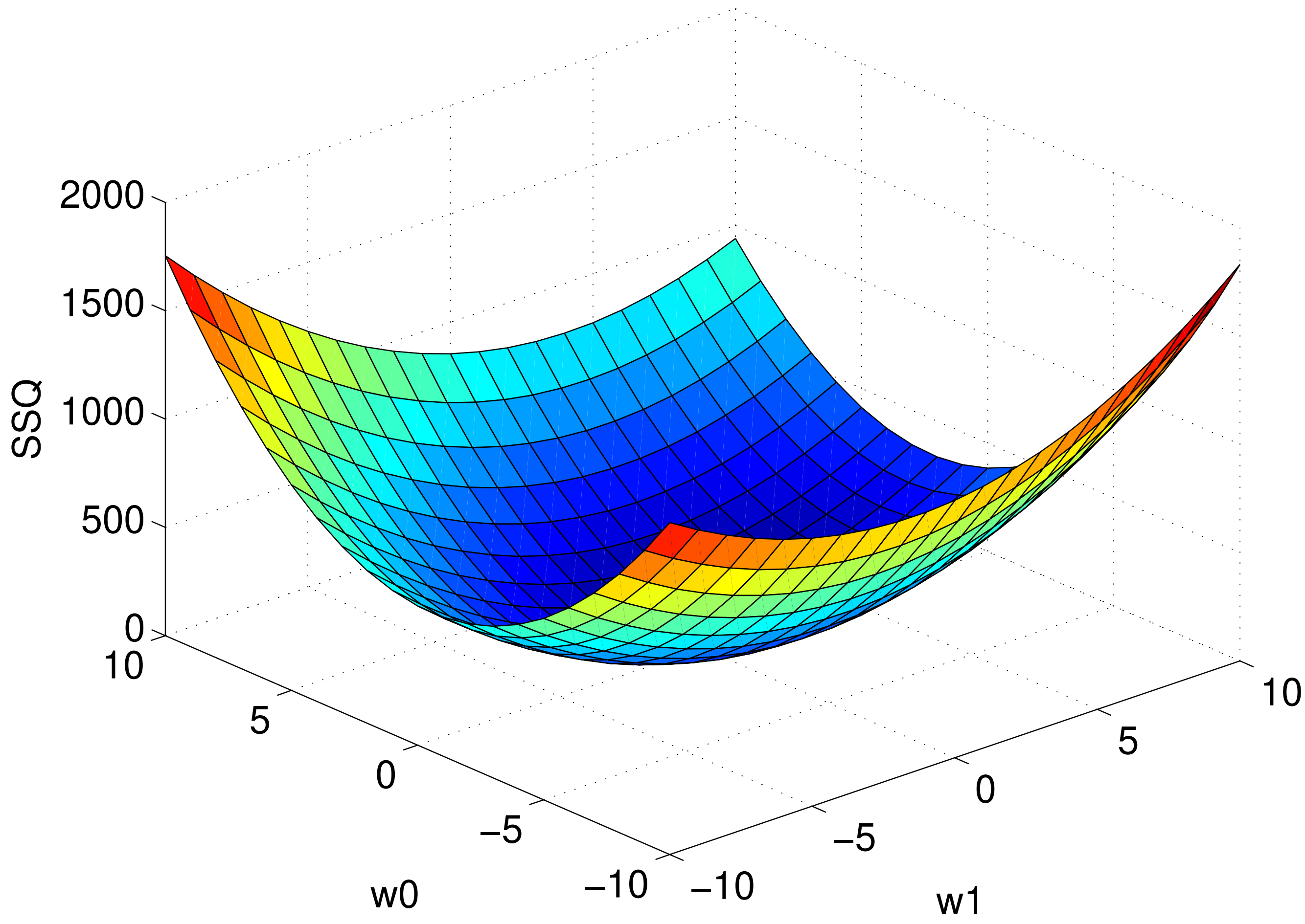
Recalling the example linear regression problem



1. Considering linear hypotheses for predicting y: $h_{\mathbf{w}}(x) = w_0 + w_1 x$
2. Goodness of fit for different weight vectors quantified by sum-squared error (SSQ): $J = \sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(x_i))^2$

w_0	0	2	1
w_1	1	2	-1
J	13.49	10.83	35.41

The error surface



- How to find $\mathbf{w}$ that minimize the SSQ?

Notation reminder

- Consider a function $f(u_1, u_2, \ldots, u_n) : \Re^n \mapsto \Re$ (for us, this will usually be an error function)
- The *partial derivative* w.r.t. u_i is denoted $\frac{\partial}{\partial u_i} f(u_1, u_2, \ldots, u_n) : \Re^n \mapsto \Re$. The partial derivative is the derivative along the u_i axis, keeping all other variables fixed.
- The *gradient* $\nabla f(u_1, u_2, \ldots, u_n) : \Re^n \mapsto \Re^n$ is a function which outputs a vector containing the partial derivatives.
  That is, $\nabla f = < \frac{\partial}{\partial u_1} f, \frac{\partial}{\partial u_2} f, \ldots, \frac{\partial}{\partial u_n} f >$.
- A necessary but not sufficient condition for $u_1, u_2, \ldots, u_n$ to be a minimum of f is that $\nabla f(u_1, u_2, \ldots u_n) = \mathbf{0}$.

A bit of algebra

$$
\begin{aligned}
\frac{\partial}{\partial w_j} J(\mathbf{w}) &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x_i}) - y_i)^2 \\
&= \frac{1}{2} \cdot 2 \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x_i}) - y_i) \frac{\partial}{\partial w_j} (h_{\mathbf{w}}(\mathbf{x_i}) - y_i) \\
&= \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x_i}) - y_i) \frac{\partial}{\partial w_j} \left(\sum_{l=0}^{n} w_l x_{i,l} - y_i \right) \\
&= \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x_i}) - y_i) x_{i,j}
\end{aligned}
$$

Setting all these partial derivatives to 0, we get a system with $(n+1)$ equations and $(n+1)$ unknowns.

The solution

- Let X be the $m \times n$ input matrix, Y the $m \times 1$ vector of outputs
- Then $J = (X\mathbf{w} - Y)^T(X\mathbf{w} - Y)$
- Recalling some multivariate calculus:

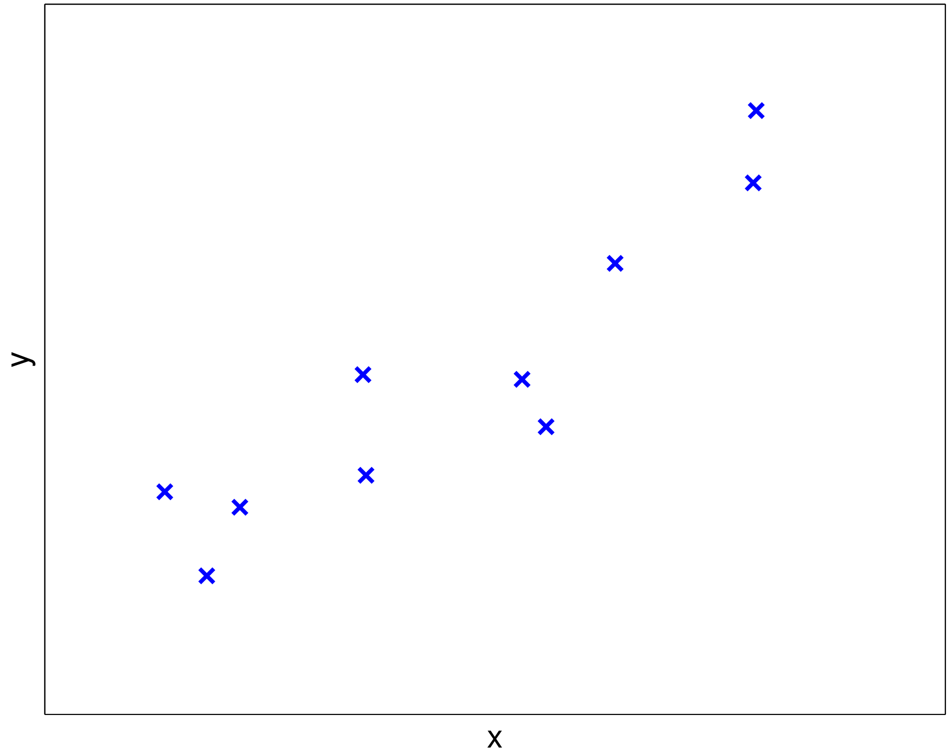
$$
\begin{aligned}
\nabla_{\mathbf{w}} J &= \nabla_{\mathbf{w}} (X\mathbf{w} - Y)^T(X\mathbf{w} - Y) \\
&= \nabla_{\mathbf{w}} (\mathbf{w}^T X^T X\mathbf{w} - Y^T X\mathbf{w} - \mathbf{w}^T X^T Y + Y^T Y) \\
&= 2X^T X\mathbf{w} - 2X^T Y
\end{aligned}
$$

- Setting gradient equal to zero:

$$
\begin{aligned}
2X^T X\mathbf{w} - 2X^T Y &= 0 \\
\Rightarrow X^T X\mathbf{w} &= X^T Y \\
\Rightarrow \mathbf{w} = (X^T X)^{-1} X^T Y
\end{aligned}
$$

- The inverse exists if the columns of X are linearly independent.

Back to our abstract example



x	y
0.86	2.49
0.09	0.83
-0.85	-0.25
0.87	3.10
-0.44	0.87
-0.43	0.02
-1.10	-0.12
0.40	1.81
-0.96	-0.83
0.17	0.43

Data matrices

$$X = \begin{bmatrix} 0.86 & 1 \\ 0.09 & 1 \\ -0.85 & 1 \\ 0.87 & 1 \\ -0.44 & 1 \\ -0.43 & 1 \\ -1.10 & 1 \\ 0.40 & 1 \\ -0.96 & 1 \\ 0.17 & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{bmatrix}$$

$$X^T X =$$

$$\begin{bmatrix} 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0.86 & 1 \\ 0.09 & 1 \\ -0.85 & 1 \\ 0.87 & 1 \\ -0.44 & 1 \\ -0.43 & 1 \\ -1.10 & 1 \\ 0.40 & 1 \\ -0.96 & 1 \\ 0.17 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4.95 & -1.39 \\ -1.39 & 10 \end{bmatrix}$$

X^TY

$$X^TY =$$

$$\left[\begin{array}{cccccccccc} 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \times \left[\begin{array}{c} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{array}\right]$$

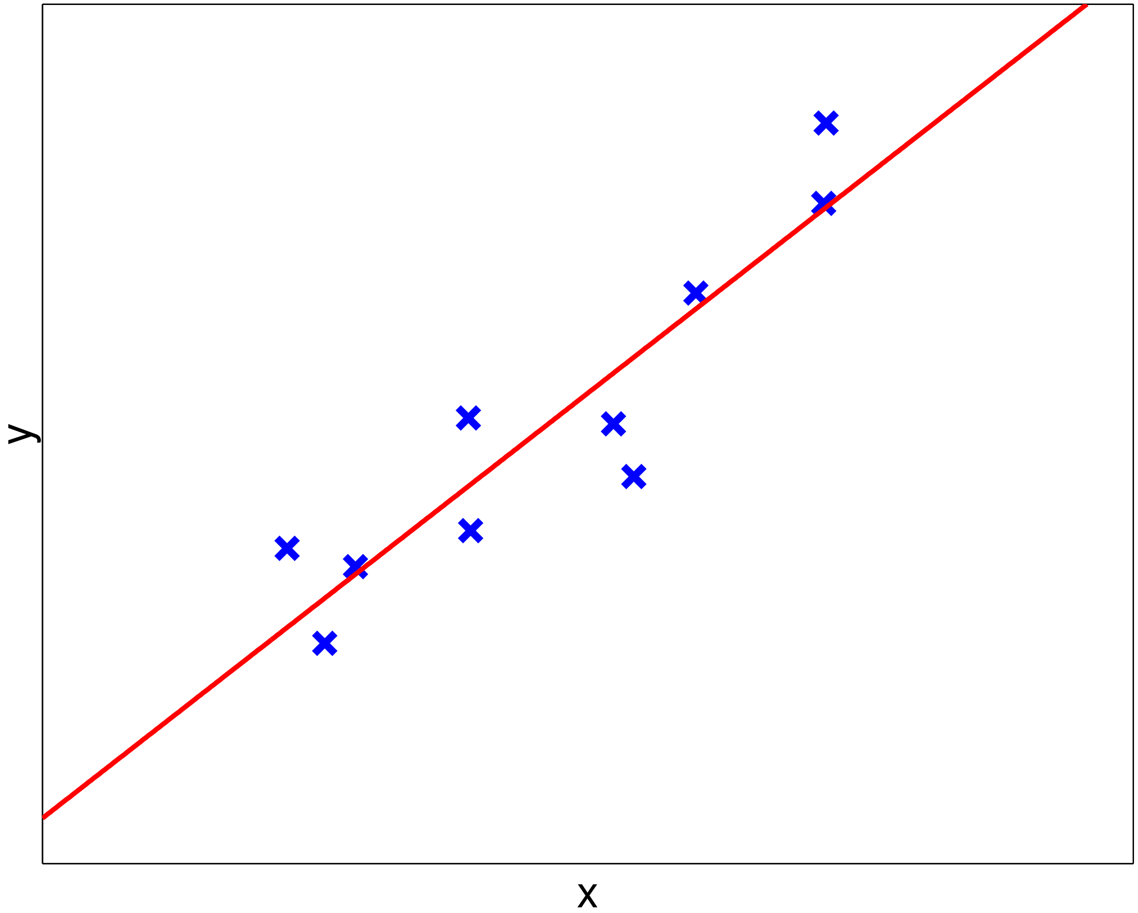
$$= \left[\begin{array}{c} 6.49 \\ 8.34 \end{array}\right]$$

Solving for $\mathbf{w}$

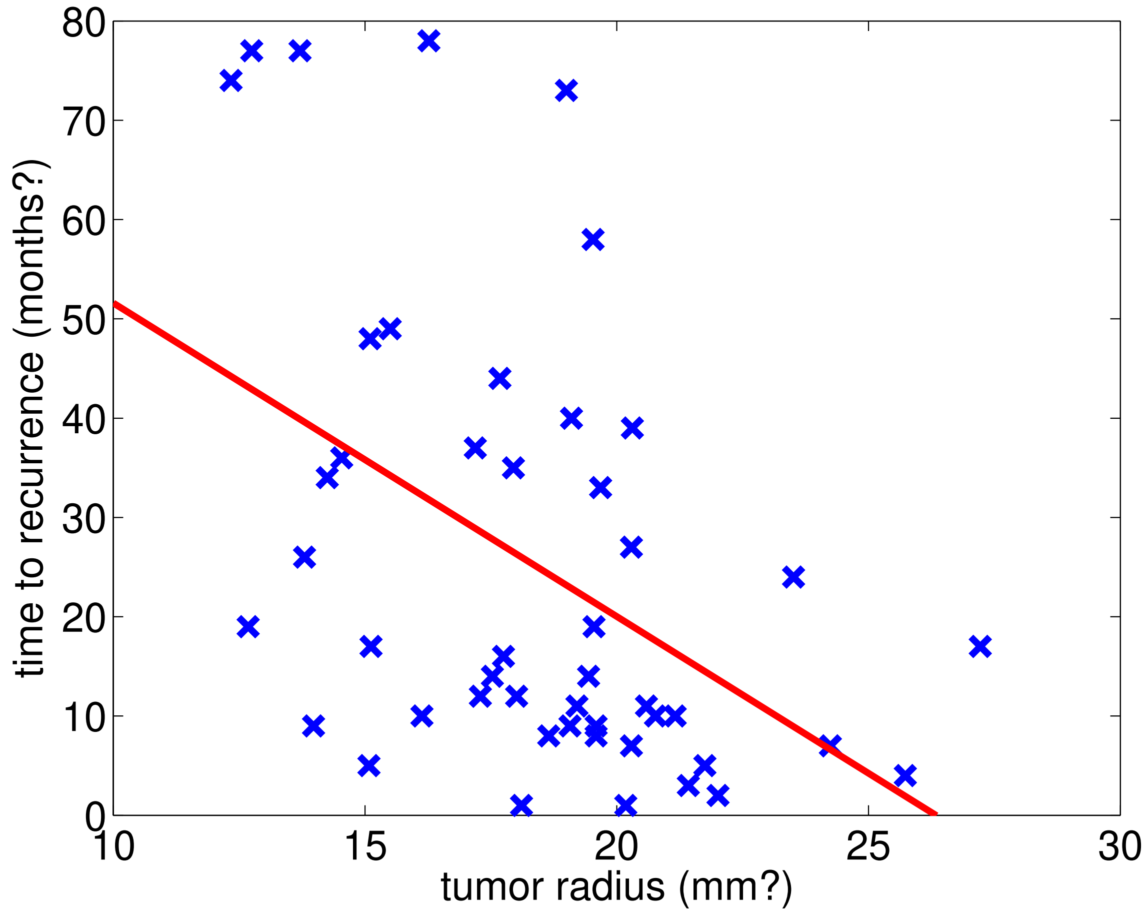
$$\mathbf{w} = (X^T X)^{-1} X^T Y = \begin{bmatrix} 4.95 & -1.39 \\ -1.39 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 6.49 \\ 8.34 \end{bmatrix} = \begin{bmatrix} 1.60 \\ 1.05 \end{bmatrix}$$

So the best fit line is $y = 1.60x + 1.05$.

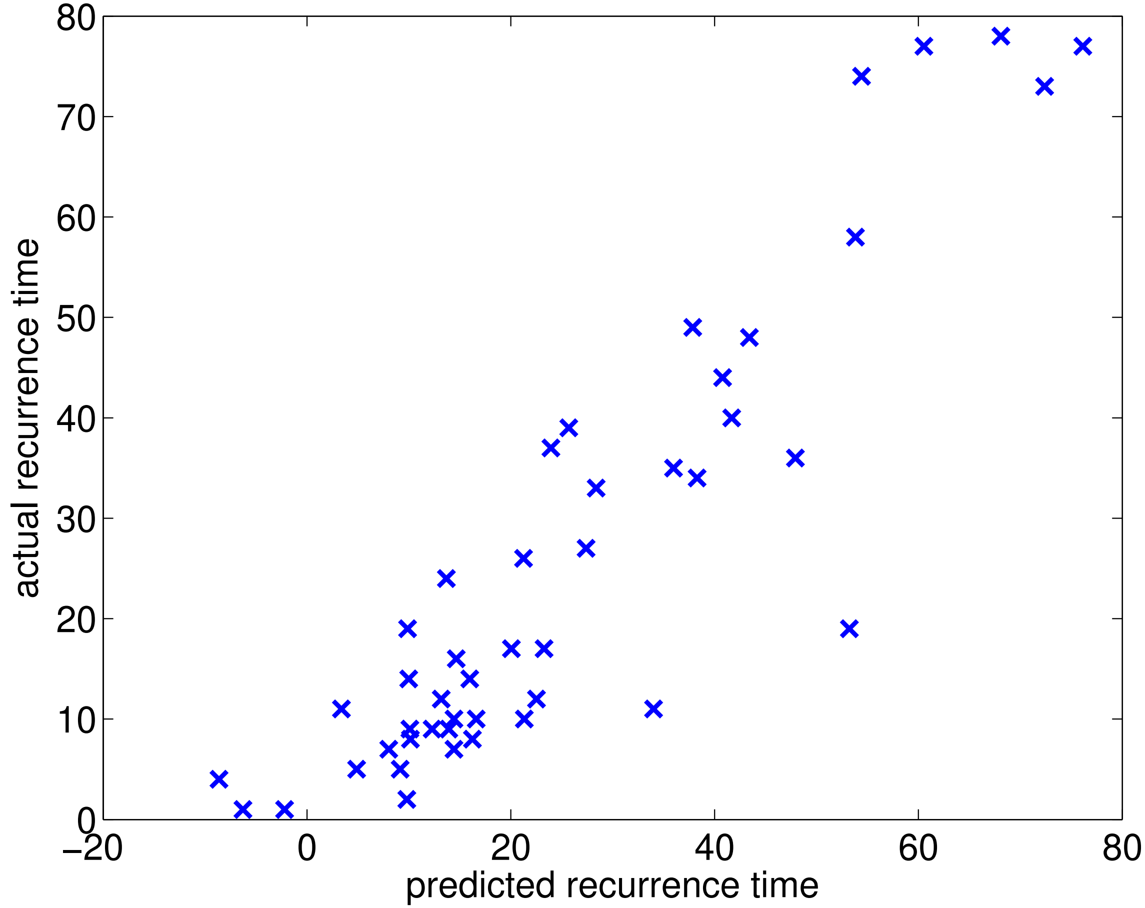
Data and line $y = 1.60x + 1.05$



Predicting recurrence time based on tumor size



Predicting recurrence time based on all input features



Supervised learning summary

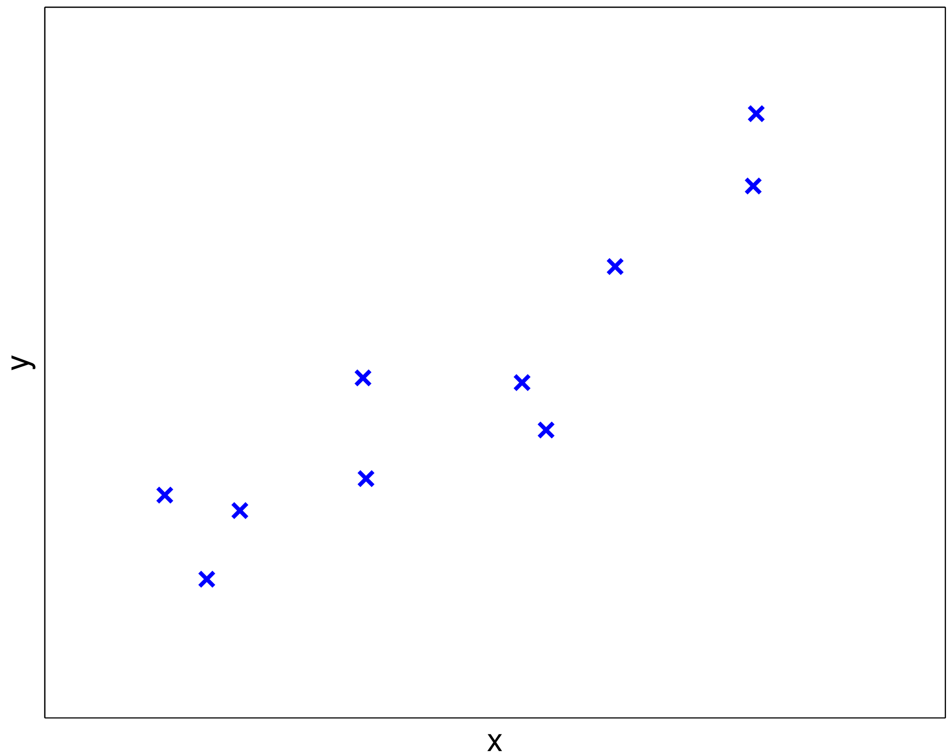
- Supervised learning is the estimation the dependence of an output variable y on input variables $\mathbf{x}$
- Supervised learning problems are often solved something like this:
  1. Data collection.
  2. Input and output variable selection and encoding
  3. Hypothesis class selection
  4. Error function selection
  5. Learning algorithm chooses "best" hypothesis

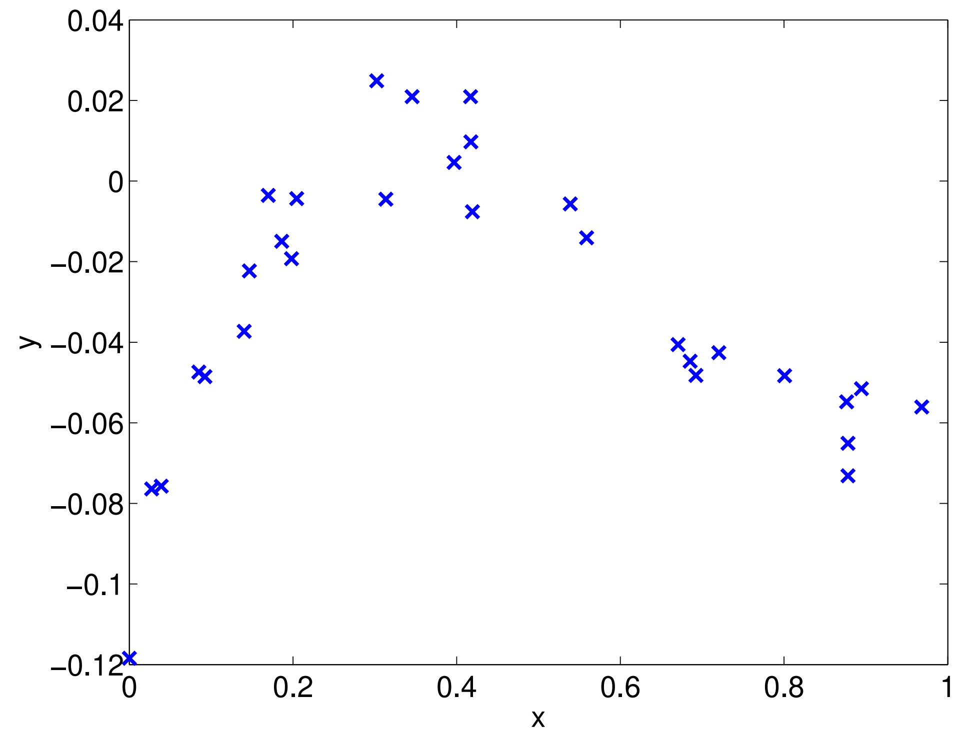
Linear regression summary

- Assumes a linear (really, affine) relationship between output variable y and input vector $\mathbf{x}$: $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^T\mathbf{w}$
- Taking sum-squared-error as defining goodness of fit, we can analytically determine the optimal weights—by setting the gradient equal to zero, and solving the resulting system of equations!
- The optimal weights can be computed in polynomial time in the size of the data set.
- The solution is $\mathbf{w} = (X^TX)^{-1}X^TY$, where X is the data matrix (augmented with a column of ones), and Y is the column vector of target outputs.

Questions on linear regression?

Is a linear hypothesis really a good choice?





- What if we want to fit a polynomial curve to the data?

The univariate polynomial hypothesis class

- We approximate the data as

$$y \approx h_{\mathbf{w}}(x) = w_d x^d + w_{d-1} x^{d-1} + \ldots + w_2 x^2 + w_1 x + w_0$$

  where d is the degree of the polynomial.
- Suppose again that we want to minimize the sum-squared error:

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2}\sum_{i=1}^{m}(h_{\mathbf{w}}(x_i) - y)^2 \\ &= \frac{1}{2}\sum_{i=1}^{m}\left(\sum_{l=0}^{d} w_l x_i^l - y\right)^2 \end{aligned}$$

- How do we find the $\mathbf{w}$ that minimize J?

Analytical solution

- Take partial derivatives of J with respect to each w_j:

$$\begin{aligned}\frac{\partial}{\partial w_j} J &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_{i=1}^{m} (h_{\mathbf{w}}(x_i) - y)^2 \\ &= \sum_{i=1}^{m} (h_{\mathbf{w}}(x_i) - y) \frac{\partial}{\partial w_j} h_{\mathbf{w}}(x_i) \\ &= \sum_{i=1}^{m} (h_{\mathbf{w}}(x_i) - y) x_i^j\end{aligned}$$

- We have a linear system of $d+1$ equations in $d+1$ unknowns (w_0, w_1, ..., w_d). Solve!

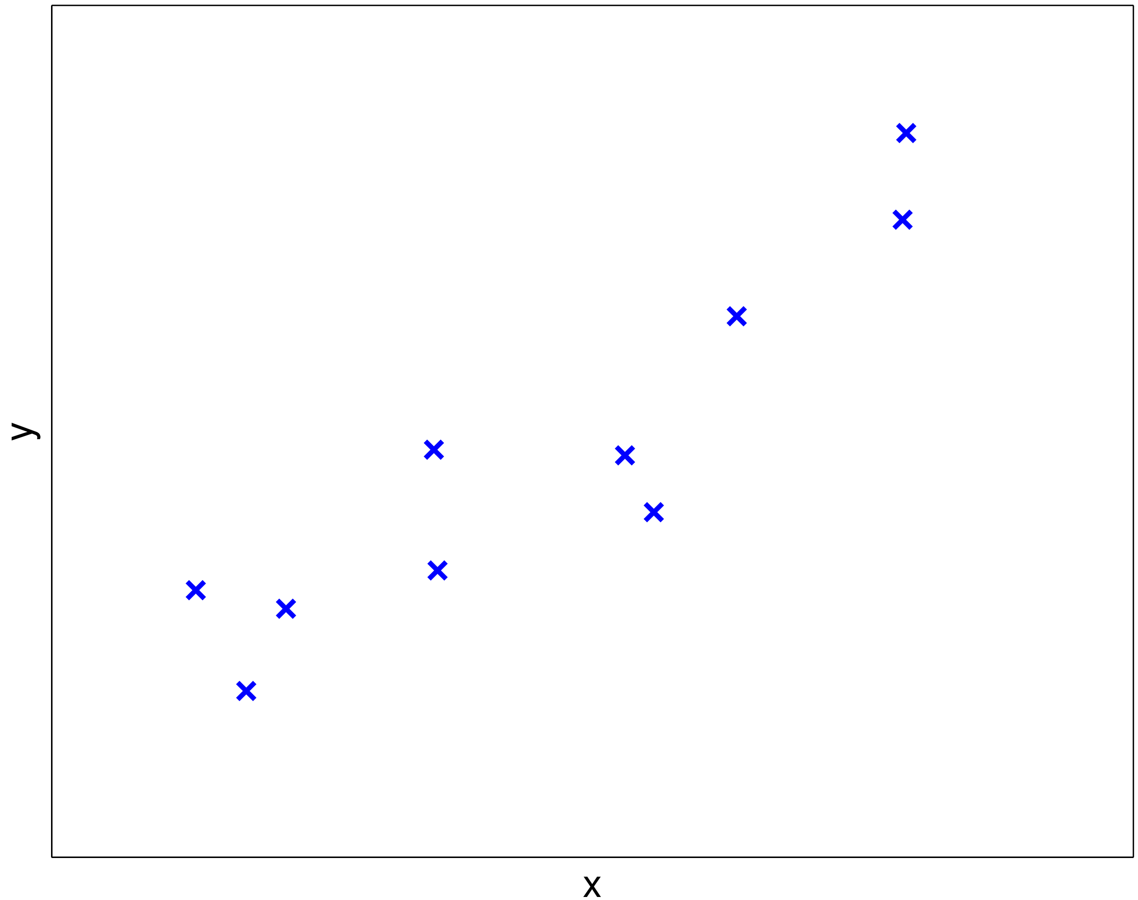
The linear algebra view

- For a degree-d fit, let X be the modified data matrix below, and Y as for linear regression:

$$X = \begin{bmatrix} x_1^d & x_1^{d-1} & \dots & x_1 & 1 \\ x_2^d & x_2^{d-1} & \dots & x_2 & 1 \\ x_3^d & x_3^{d-1} & \dots & x_3 & 1 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ x_m^d & x_m^{d-1} & \dots & x_m & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_m \end{bmatrix}$$

- Then $J(w) = (X\mathbf{w} - Y)^T(X\mathbf{w} - Y)$, and the previous analysis for linear regression follows.
- The optimal weights are $\mathbf{w} = (X^TX)^{-1}X^TY$.

Recalling our abstract regression problem



- Suppose we seek an order-2 fit: $y \approx w_2 x^2 + w_1 x + w_0$.

Example of quadratic regression: Data matrices

$$X = \begin{bmatrix} 0.75 & 0.86 & 1 \\ 0.01 & 0.09 & 1 \\ 0.73 & -0.85 & 1 \\ 0.76 & 0.87 & 1 \\ 0.19 & -0.44 & 1 \\ 0.18 & -0.43 & 1 \\ 1.22 & -1.10 & 1 \\ 0.16 & 0.40 & 1 \\ 0.93 & -0.96 & 1 \\ 0.03 & 0.17 & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{bmatrix}$$

$$X^T X =$$

$$\begin{bmatrix} 0.75 & 0.01 & 0.73 & 0.76 & 0.19 & 0.18 & 1.22 & 0.16 & 0.93 & 0.03 \\ 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0.75 & 0.86 & 1 \\ 0.01 & 0.09 & 1 \\ 0.73 & -0.85 & 1 \\ 0.76 & 0.87 & 1 \\ 0.19 & -0.44 & 1 \\ 0.18 & -0.43 & 1 \\ 1.22 & -1.10 & 1 \\ 0.16 & 0.40 & 1 \\ 0.93 & -0.96 & 1 \\ 0.03 & 0.17 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4.11 & -1.64 & 4.95 \\ -1.64 & 4.95 & -1.39 \\ 4.95 & -1.39 & 10 \end{bmatrix}$$

X^TY

$$X^TY =$$

$$\begin{bmatrix} 0.75 & 0.01 & 0.73 & 0.76 & 0.19 & 0.18 & 1.22 & 0.16 & 0.93 & 0.03 \\ 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{bmatrix}$$

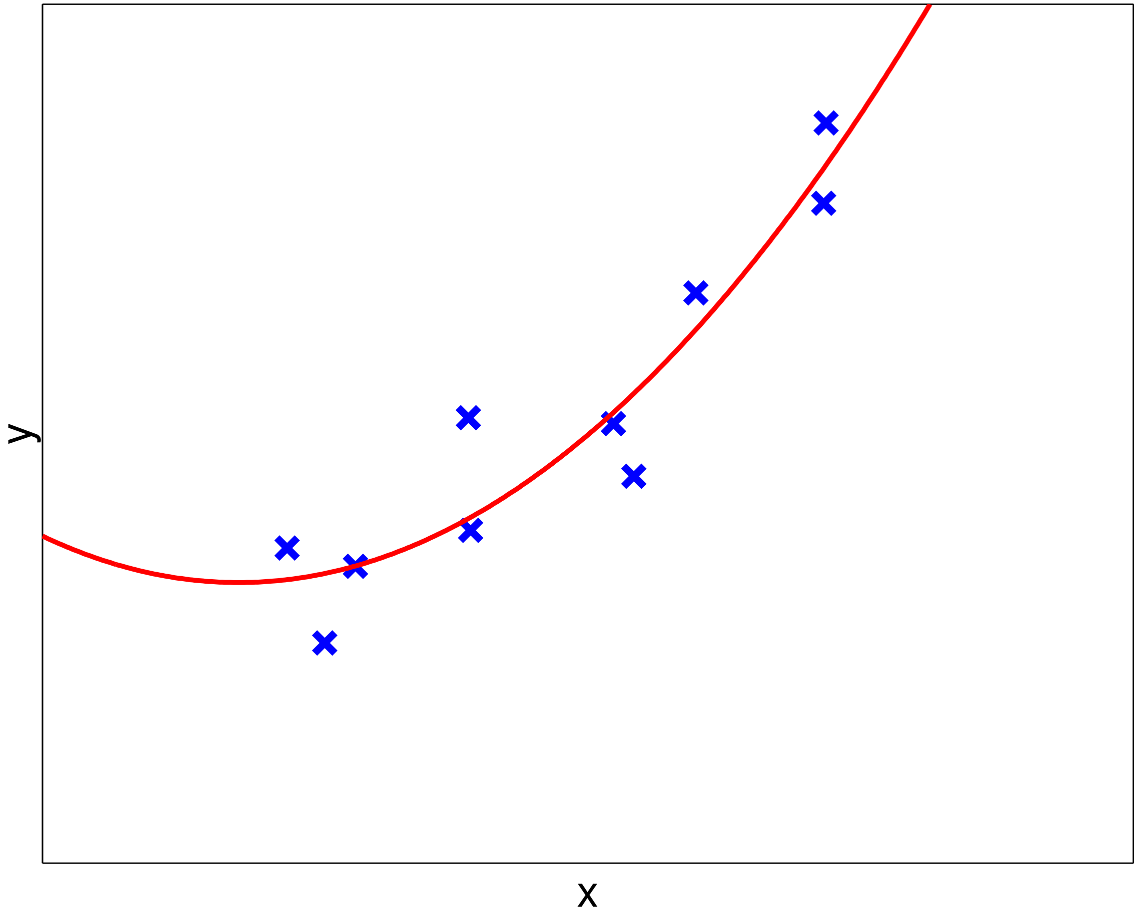
$$= \begin{bmatrix} 3.60 \\ 6.49 \\ 8.34 \end{bmatrix}$$

Solving for $\mathbf{w}$

$$\mathbf{w} = (X^T X)^{-1} X^T Y = \begin{bmatrix} 4.11 & -1.64 & 4.95 \\ -1.64 & 4.95 & -1.39 \\ 4.95 & -1.39 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 3.60 \\ 6.49 \\ 8.34 \end{bmatrix} = \begin{bmatrix} 0.68 \\ 1.74 \\ 0.73 \end{bmatrix}$$

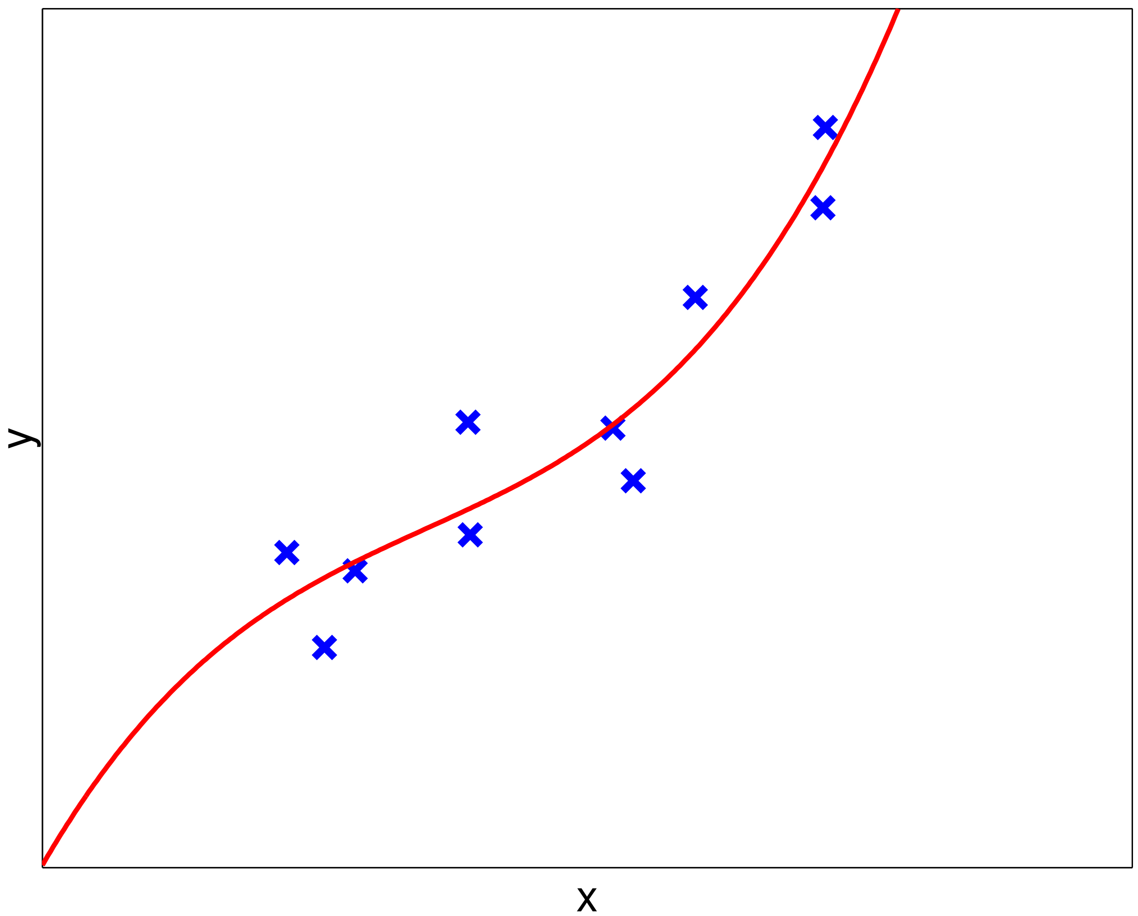
So the best order-2 polynomial is $y = 0.68x^2 + 1.74x + 0.73$.

Data and curve $y = 0.68x^2 + 1.74x + 0.73$



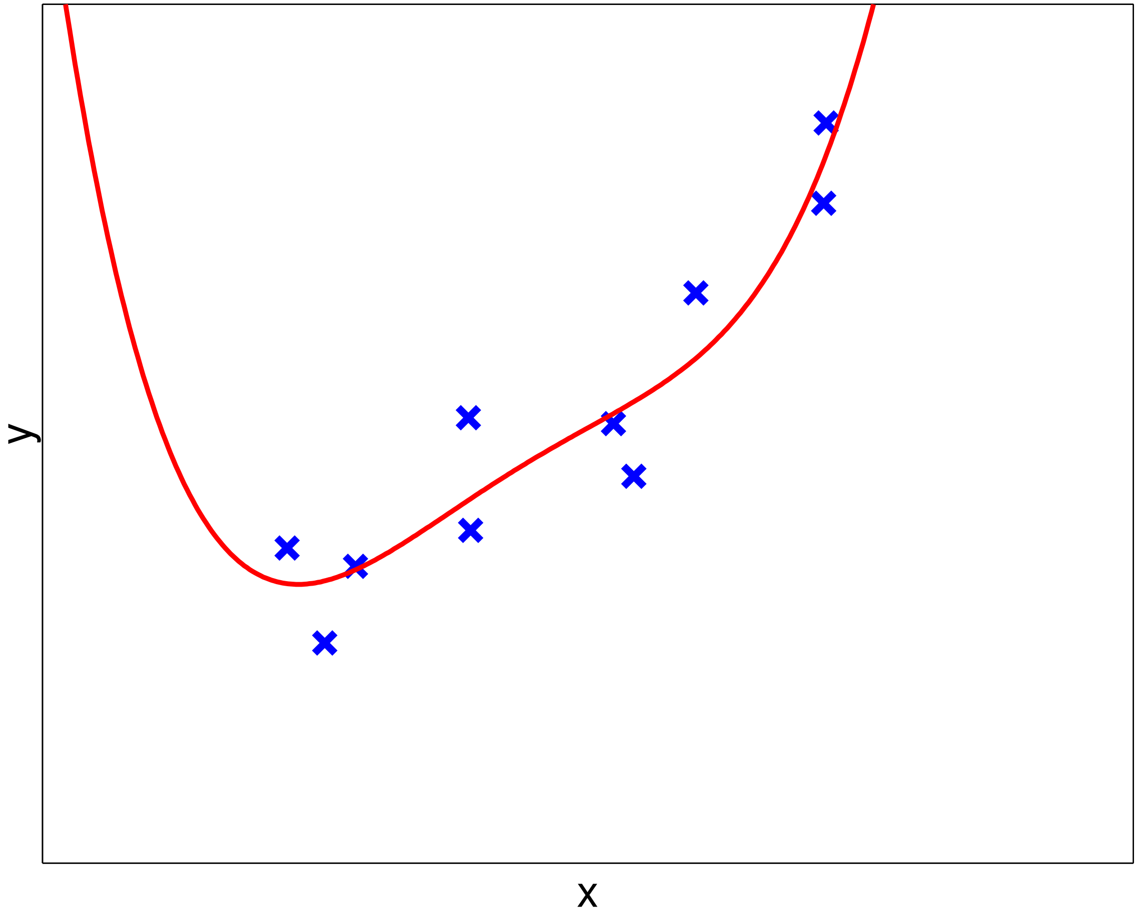
Is this a better fit to the data?

Order-3 fit



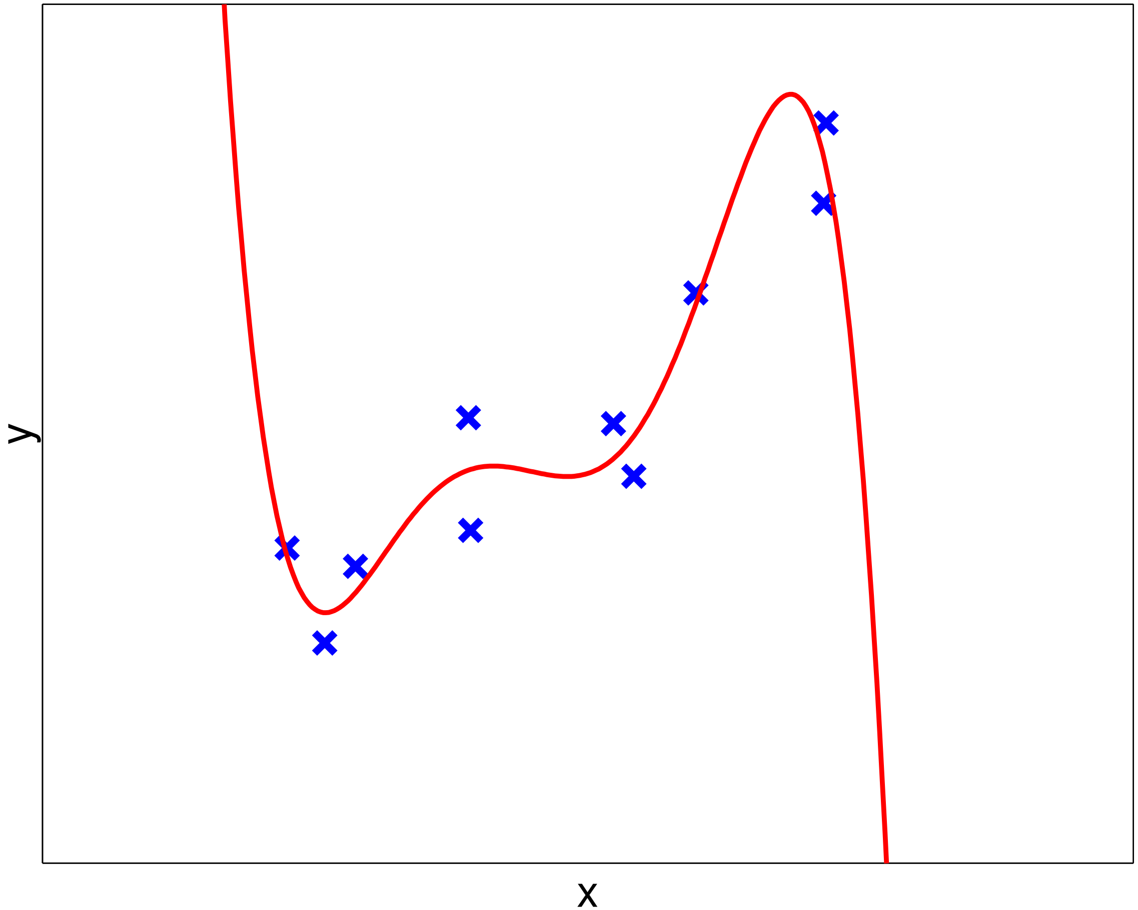
Is this a better fit to the data?

Order-4 fit



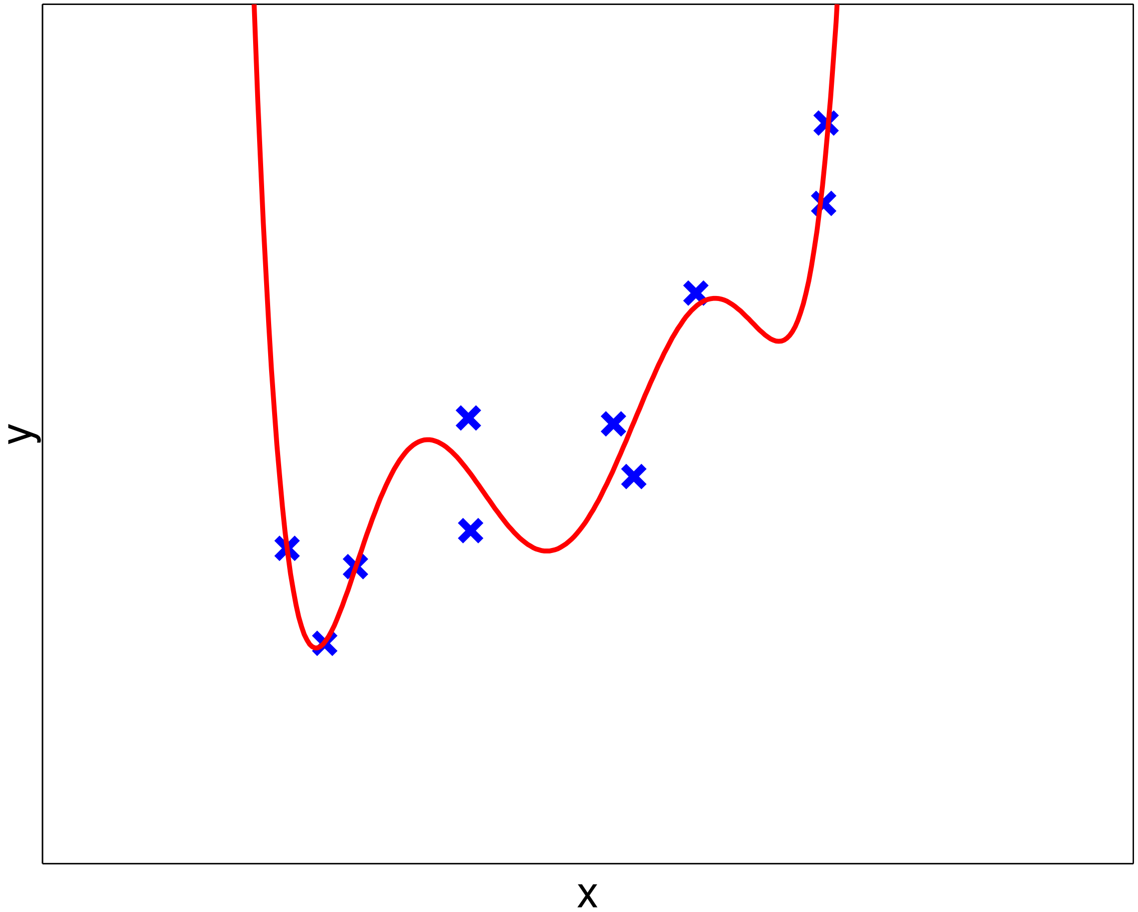
Is this a better fit to the data?

Order-5 fit



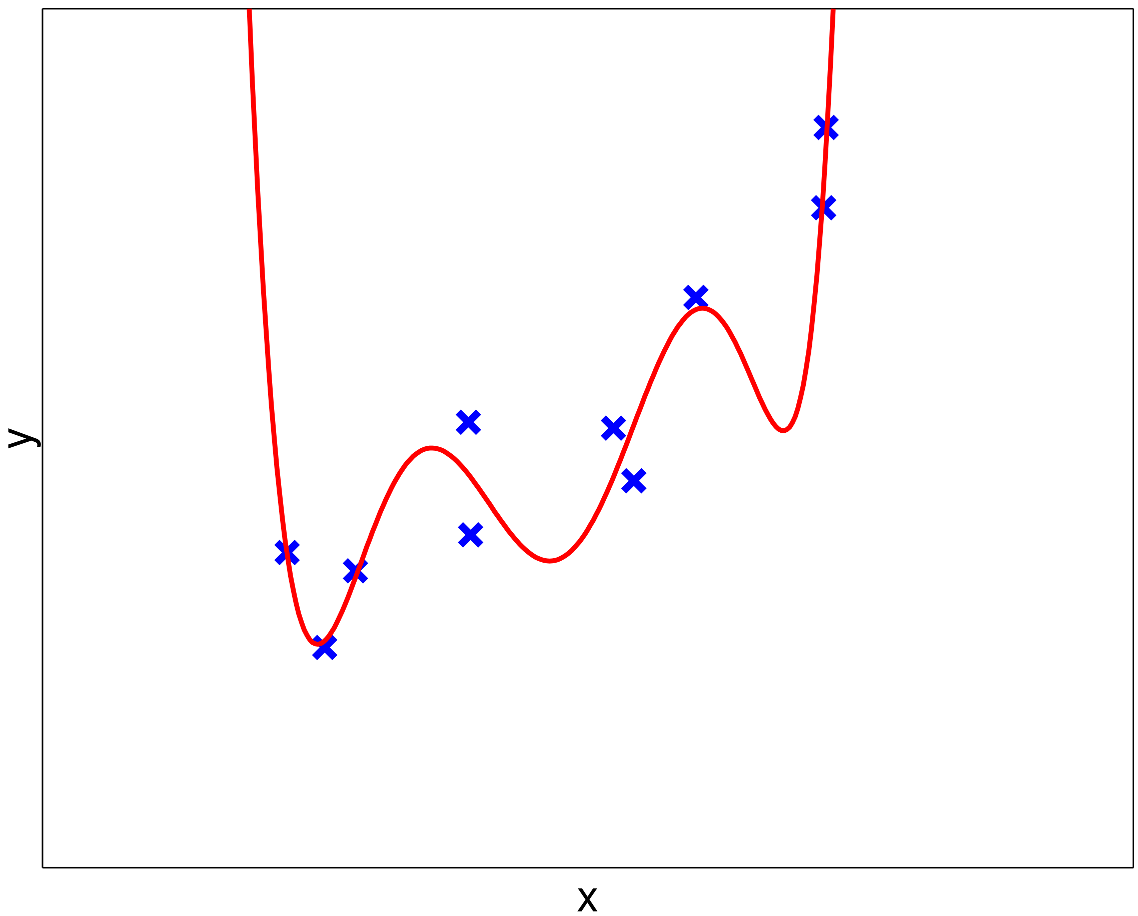
Is this a better fit to the data?

Order-6 fit



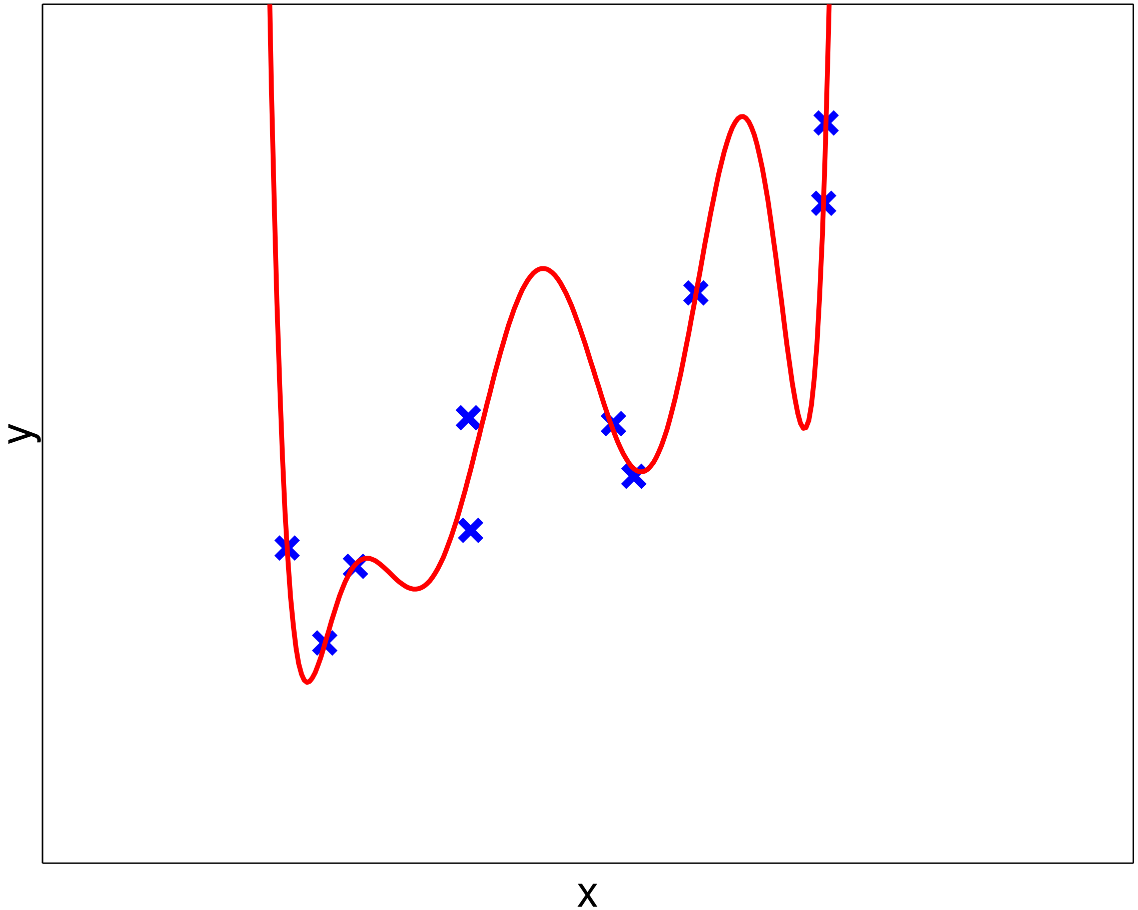
Is this a better fit to the data?

Order-7 fit



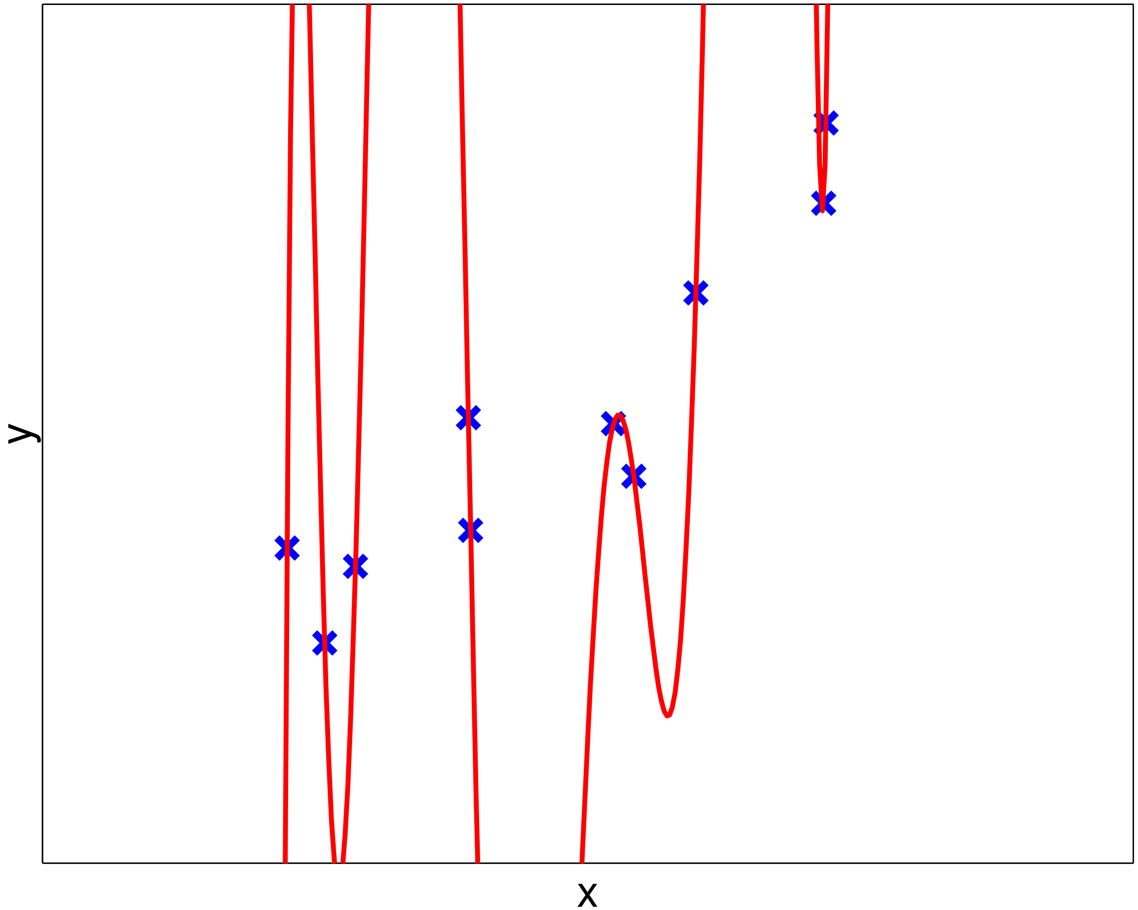
Is this a better fit to the data?

Order-8 fit



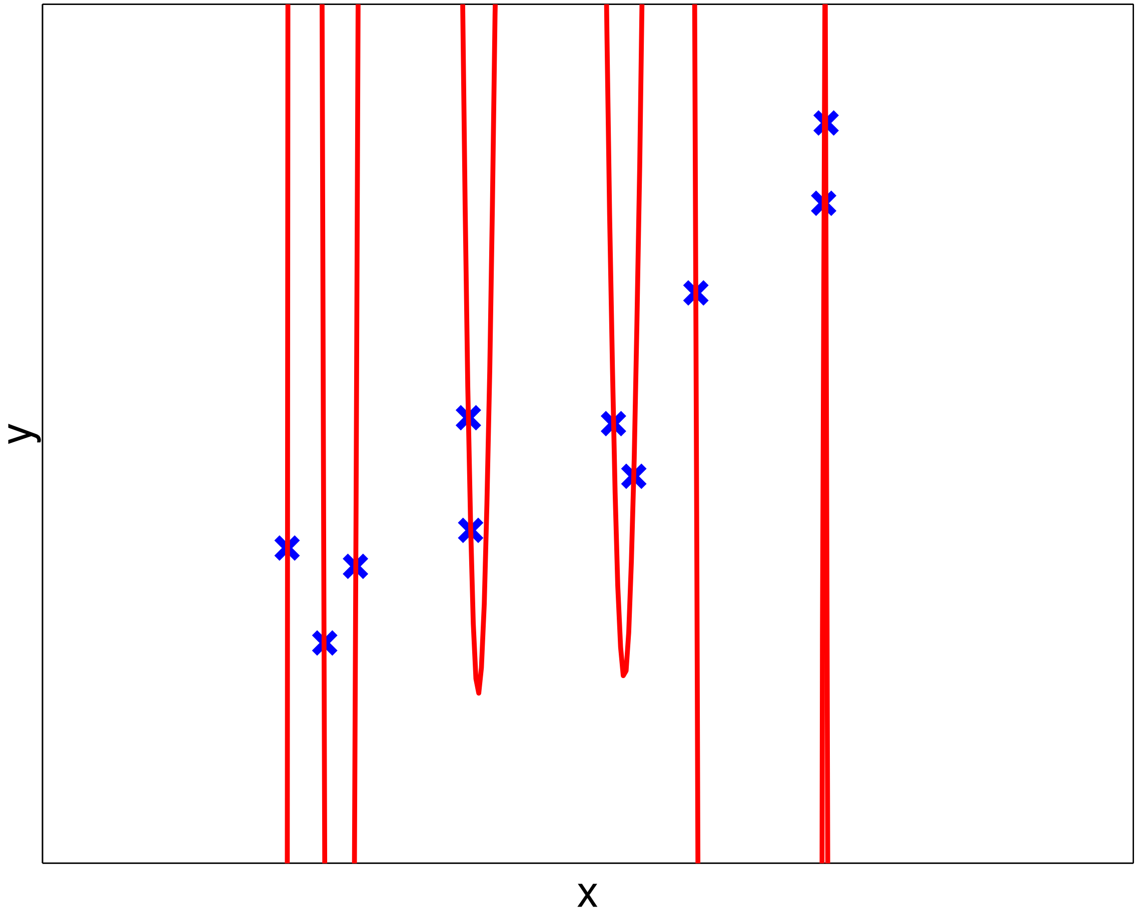
Is this a better fit to the data?

Order-9 fit



Is this a better fit to the data?

Order-10 fit



Is this a better fit to the data?

Overfitting

- A general, **HUGELY IMPORTANT** problem for all machine learning algorithms
- We can find a hypothesis that predicts perfectly the training data but **does not generalize** well to new data
- E.g., a lookup table!
- We are seeing an instance here: if we have a lot of parameters, the hypothesis "memorizes" the data points, but is wild everywhere else.

Overfitting more formally

- Every hypothesis has a "true" error $J^*(h)$ (measured on all possible data items we could ever encounter)
- Because we do not have all the data, we measure the error on the training set $J_D(h)$
- Suppose we compare hypotheses h_1 and h_2 on the training set, and $J_D(h_1) < J_D(h_2)$
- If h_2 is "truly" better, i.e. $J^*(h_2) < J^*(h_1)$, our algorithm is overfitting.
- We need theoretical and empirical methods to guard against it!

Leave-one-out cross-validation

- How can we choose the best d for an order-d polynomial fit to the data?
- Repeat the following procedure:
  - Leave out *one instance* from the training set, to estimate the true prediction error for the best order-d fit for $d \in \{1, 2, \ldots, 9\}$.
  - Use all the other data items for finding $\mathbf{w}$
  - Measure the error on the instance left out
  - This is an estimate of the *true prediction error*
- Choose the d with lowest average prediction error

Cross-validation, more generally

- A general procedure for estimating the true error of a predictor
- The data is split into three subsets:
  - A *training set* used only to find the parameters $\mathbf{w}$
  - A *validation set* used to find the right hypothesis class (e.g. the degree of the polynomial)
  - A *test set* used to report the prediction error of the algorithm
- These set must be *disjoint*!
- The process is repeated several times, and the results are averaged to provide error estimates.

Estimating true error for $d = 1$, using leave-one-out X-val

$D = \{(0.86, 2.49), (0.09, 0.83), (-0.85, -0.25), (0.87, 3.10), (-0.44, 0.87),$
$(-0.43, 0.02), (-1.10, -0.12), (0.40, 1.81), (-0.96, -0.83), (0.17, 0.43)\}.$

Iter	D_{train}	D_{valid}	Error$_{\text{train}}$	Error$_{\text{valid}}$
1	$D - \{(0.86, 2.49)\}$	$(0.86, 2.49)$	0.4928	0.0044
2	$D - \{(0.08, 0.83)\}$	$(0.09, 0.83)$	0.1995	0.1869
3	$D - \{(-0.85, -0.25)\}$	$(-0.85, -0.25)$	0.3461	0.0053
4	$D - \{(0.87, 3.10)\}$	$(0.87, 3.10)$	0.3887	0.8681
5	$D - \{(-0.44, 0.87)\}$	$(-0.44, 0.87)$	0.2128	0.3439
6	$D - \{(-0.43, 0.02)\}$	$(-0.43, 0.02)$	0.1996	0.1567
7	$D - \{(-1.10, -0.12)\}$	$(-1.10, -0.12)$	0.5707	0.7205
8	$D - \{(0.40, 1.81)\}$	$(0.40, 1.81)$	0.2661	0.0203
9	$D - \{(-0.96, -0.83)\}$	$(-0.96, -0.83)$	0.3604	0.2033
10	$D - \{(0.17, 0.43)\}$	$(0.17, 0.43)$	0.2138	1.0490
		mean:	0.2188	0.3558

Cross-validation results

d	$\text{Error}_{\text{train}}$	$\text{Error}_{\text{valid}}$
1	0.2188	0.3558
2	0.1504	0.3095
3	0.1384	0.4764
4	0.1259	1.1770
5	0.0742	1.2828
6	0.0598	1.3896
7	0.0458	38.819
8	0.0000	6097.5
9	0.0000	6097.5

- Optimal choice: $d = 2$. Overfitting for $d > 2$
- Why are $d = 8$ and $d = 9$ the same?

Summary of (univariate) polynomial regression

- Univariate polynomial regression is solved similarly to linear regression, but with modified data matrix.
  (Multivariate polynomial regression can be handled similarly.)
- One way to obtain a nonlinear fit to the data, is to:
  1. Nonlinearly transform the input space
  2. Do a linear fit in the transformed space
- Overfitting can result when our hypothesis class is “too complex” for the data, and must be guarded against!
- Cross-validation is a general procedure that helps us protect against overfitting

Questions?

Why minimize SSQ?

Why not another error function?
What error functions could we imagine?

What is good about SSQ?

- SSQ penalizes large deviations
- SSQ is differentiable
- . . . ?

Justifying SSQ via the Principle of Maximum Likelihood

- Suppose we have a regression data set D and hypothesis space $\mathcal{H}$
- The Principle of Maximum Likelihood says that:

  "The best hypothesis is the one under which the data is most likely."

- What does that mean?
  - Typically, no $h \in \mathcal{H}$ exactly fits the data.
  - (And we may not want to do that even if we could.)

A probabilistic assumption

- For any $h \in \mathcal{H}$, we could imagine that y_i is a noisy target value, generated from a hypothesis h
- More specifically, for any assume thatt:

$$y_i = h(\mathbf{x_i}) + e_i$$

  where e_i is random variable (noise).
- The e_i might represent measurement noise or the influence of other unobserved/unmodeled processes.
- The e_i are random variables. As such, we can now conceive the probability of the data for any hypothesis: $P(D|h)$.

Maximum likelihood estimation

- Define the likelihood of hypothesis h as $l(h) = P(D|h)$
- The Principle of Maximum Likelihood states that the best hypothesis is:

$$h_{ml} = \arg\max_{\mathcal{H}} P(D|h)$$

- Standard assumption: the training examples are *independent of each other (i.i.d. samples)*
- This allows us to simplify $P(D|h)$:

$$P(D|h) = \prod_{i=1}^{m} P(\langle \mathbf{x_i}, y_i \rangle | h) = \prod_{i=1}^{m} P(y_i | \mathbf{x_i}; h)$$

(If we imagine the $\mathbf{x_i}$ are chosen deterministically.)

The $\log$ trick

- We want to maximize:

$$L(h) = \prod_{i=1}^{m} P(y_i|\mathbf{x_i}; h)$$

  This is a product, and products are hard to maximize!

- Instead, we will maximize $\log L(h)$! (the log-likelihood function)

$$\log L(h) = \sum_{i=1}^{m} \log P(y_i|\mathbf{x_i}; h)$$

Gaussian noise assumption

- Adopt the assumption that:

$$y_i = h(\mathbf{x_i}) + e_i,$$

where e_i are normally distributed with mean 0 and variance σ

- Hence,

$$L(h) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{y_i - h(\mathbf{x_i})}{\sigma}\right)^2}$$

because the noise variables e_i are from a Gaussian distribution

Applying the $\log$ trick

$$
\begin{aligned}
\log L(h) &= \sum_{i=1}^{m} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i - h(\mathbf{x_i}))^2}{\sigma^2}}\right) \\
&= \sum_{i=1}^{m} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \sum_{i=1}^{m} \frac{1}{2}\frac{(y_i - h(\mathbf{x_i}))^2}{\sigma^2}
\end{aligned}
$$

Maximizing the right hand side is the same as minimizing:

$$
\sum_{i=1}^{m} \frac{1}{2}\frac{(y_i - h(\mathbf{x_i}))^2}{\sigma^2} \propto \sum_{i=1}^{m} \frac{1}{2}(y_i - h(\mathbf{x_i}))^2
$$

This is our old friend, the sum-squared-error function!

Maximum likelihood hypothesis for least-squares estimators

- Under the assumption that the training examples are i.i.d. and that we have *Gaussian target noise*, the maximum likelihood hypothesis is that minimizing the sum squared error:

$$h_{ml} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{m} (y_i - h(\mathbf{x_i}))^2$$

- This makes explicit the hypothesis behind minimizing the sum-squared error
- If we further assume $h_{\mathbf{w}}$ to be linear, we obtain least-squares linear regression
- Next lecture we will use a similar approach for classification