# Machine Learning (COMP-652 and ECSE-608)

Lecturers: Audrey Durand and Riashat Islam
Email: audrey.durand@mcgill.ca - riashat.islam@mail.mcgill.ca

Teaching assistants: Koustuv Sinha, Pierre Thodoroff, Nishanth Anand
Emails: {koustuv.sinha,pierre.thodoroff,nishanth.anand}@mail.mcgill.ca

**Class web page:** https://rllabmcgill.github.io/COMP-652

# Course Content

- Introduction, linear models, regularization and Bayesian interpretation (3 lectures)
- Regression, kernels and Gaussian processes (3 lectures)
- Neural networks and deep learning (3 lectures)
- Latent variables and unsupervised learning (1 lecture)
- Clustering and dimensionality reduction (1 lecture)
- Reinforcement learning and Markov processes (5 lectures)
- Active learning and Bayesian optimization (1 lecture)
- Probabilistic graphical models (1 lecture)
- Optimization (1 lecture)
- Inference (2 lectures)
- Generative models (1 lecture)

# Today's Lecture Outline

- Administrative issues
- What is machine learning?
- Types of machine learning
- Linear hypotheses
- Error functions
- Overfitting

# Administrative issues

- Class materials:
  - No required textbook, but several textbooks available
  - Required or recommended readings (from books or research papers) posted on the class web page
  - Class notes: posted on the web page
- Prerequisites:
  - Knowledge of a programming language
  - Knowledge of probability/statistics, calculus and linear algebra; general facility with math
  - Some AI background is recommended *but not required*[1]

[1] show of hands: who took COMP-424? COMP-551?

# Office hours

- Right after class: 4pm-5pm Mondays and Wednesdays (one of the lecturers).
- 10am-11am Tuesdays (Pierre), 11am-12pm Wednesdays (Nishanth), Thursdays 3pm-4pm (Koustuv).
- Meeting with any of the lecturers or TAs at other times
  by appointments only.

# Evaluation

- Three homework assignments (30%)
- Project (30%)
- Final examination (40%)
- Participation to class discussions (up to 1% extra credit)

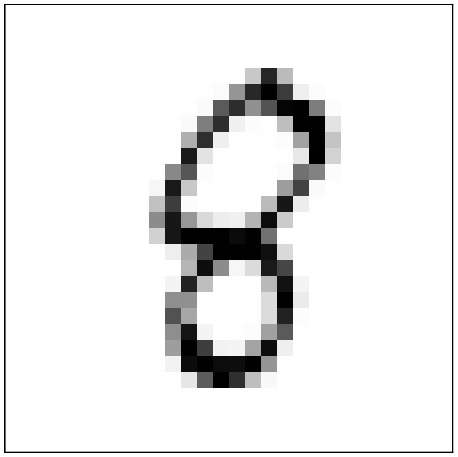
# What is machine learning?

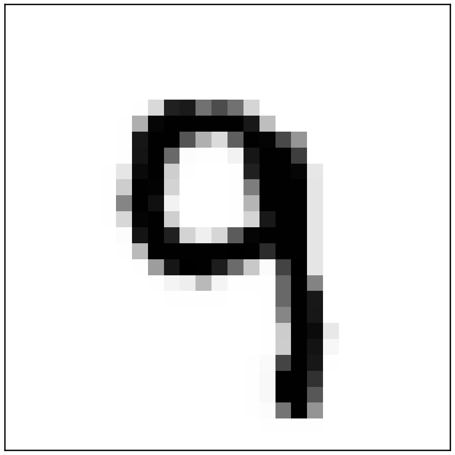
- A. Samuel, who coined the term *machine learning*:

  "Field of study that gives computers the ability to learn without being explicitly programmed." [*Some Studies in Machine Learning Using the Game of Checkers*, 1959]

# What is learning?

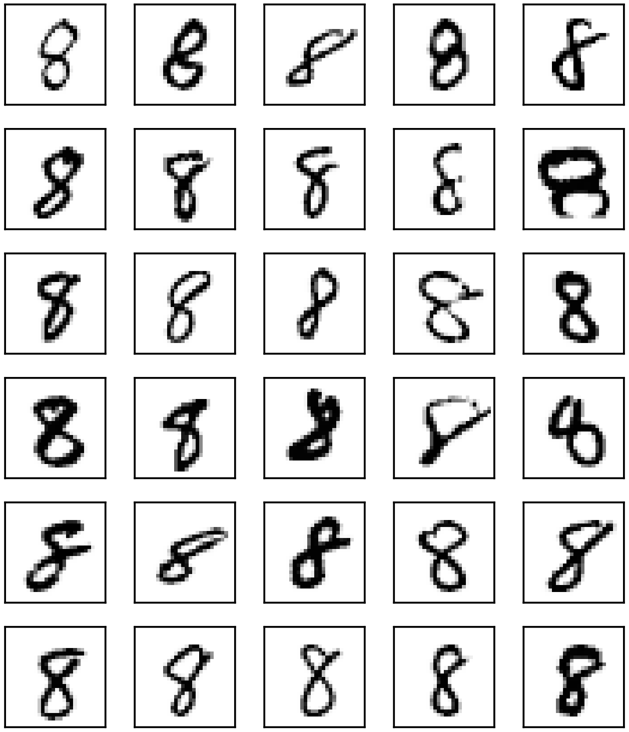
- Let's say you want to write a program distinguishing images of the digits 8 and 9...

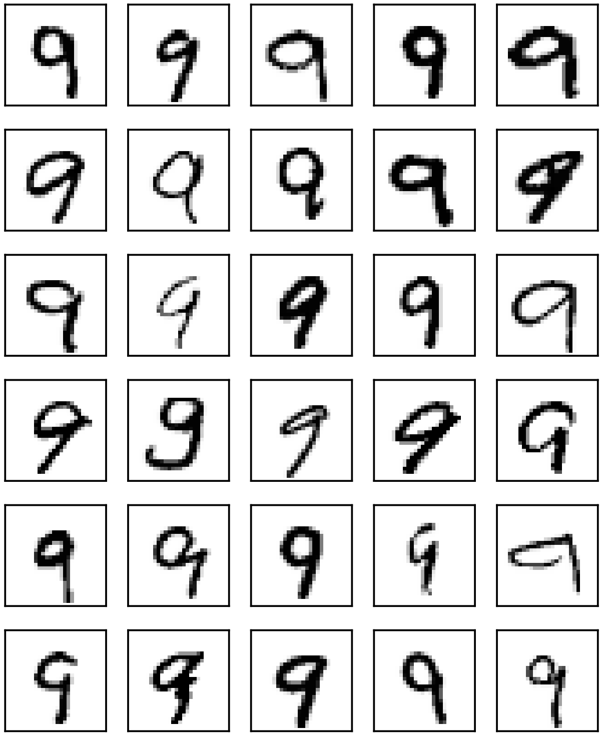




# What is learning?

- The Machine Learning way:





# What is learning?

- A. Samuel, who coined the term *machine learning*:

  "Field of study that gives computers the ability to learn without being explicitly programmed." [*Some Studies in Machine Learning Using the Game of Checkers*, 1959]

- H. Simon: Any process by which a system improves its performance

  "Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the task or tasks drawn from the same population more efficiently and more effectively the next time." [*Machine Learning I*, 1983]

- M. Minsky:

  "Learning is making *useful* changes in our minds." [*The Society of Mind*, 1985]

- R. Michalski:

  "Learning is constructing or modifying representations of what is being experienced." [Understanding the Nature of Learning, 1986]

# Why study machine learning?

Engineering reasons:

- Easier to build a learning system than to hand-code a working program! E.g.:
  - Robot that learns a map of the environment by exploring
  - Programs that learn to play games by playing against themselves
- Improving on existing programs, e.g.
  - Instruction scheduling and register allocation in compilers
  - Combinatorial optimization problems
- Solving tasks that require a system to be adaptive, e.g.
  - Speech and handwriting recognition
  - "Intelligent" user interfaces

# Why study machine learning?

Scientific reasons:

- Discover knowledge and patterns in highly dimensional, complex data
  - Sky surveys
  - High-energy physics data
  - Sequence analysis in bioinformatics
  - Social network analysis
  - Ecosystem analysis
- Understanding animal and human learning
  - How do we learn language?
  - How do we recognize faces?
- Creating real AI!

  "If an expert system–brilliantly designed, engineered and implemented–cannot learn not to repeat its mistakes, it is not as intelligent as a worm or a sea anemone or a kitten." (Oliver Selfridge).

# Very brief history

- Studied ever since computers were invented (e.g. Samuel's checkers player, 1959)
- Very active in 1960s (neural networks)
- Died down in the 1970s
- Revival in early 1980s (decision trees, backpropagation, temporal-difference learning) - coined as "machine learning"
- Exploded starting in the 1990s
- Now: very active research field, several yearly conferences (e.g., ICML, NIPS), major journals (e.g., Machine Learning, Journal of Machine Learning Research), rapidly growing number of researchers
- The time is right to study in the field!
  - Lots of recent progress in algorithms and theory
  - Flood of data to be analyzed
  - Computational power is available
  - Growing demand for industrial applications

# What are good machine learning tasks?

- There is no human expert

  E.g., DNA analysis
- Humans can perform the task but cannot explain how

  E.g., character recognition
- Desired function changes frequently

  E.g., predicting stock prices based on recent trading data
- Each user needs a customized function

  E.g., news filtering

# Important application areas

- Bioinformatics: sequence alignment, analyzing microarray data, information integration, ...
- Computer vision: object recognition, tracking, segmentation, active vision, ...
- Robotics: state estimation, map building, decision making
- Graphics: building realistic simulations
- Speech: recognition, speaker identification
- Financial analysis: option pricing, portfolio allocation
- E-commerce: automated trading agents, data mining, spam, ...
- Medicine: diagnosis, treatment, drug design,...
- Computer games: building adaptive opponents
- Multimedia: retrieval across diverse databases

# Kinds of learning

Based on the information available:

- Supervised learning
- Reinforcement learning
- Unsupervised learning

Based on the role of the learner

- Passive learning
- Active learning

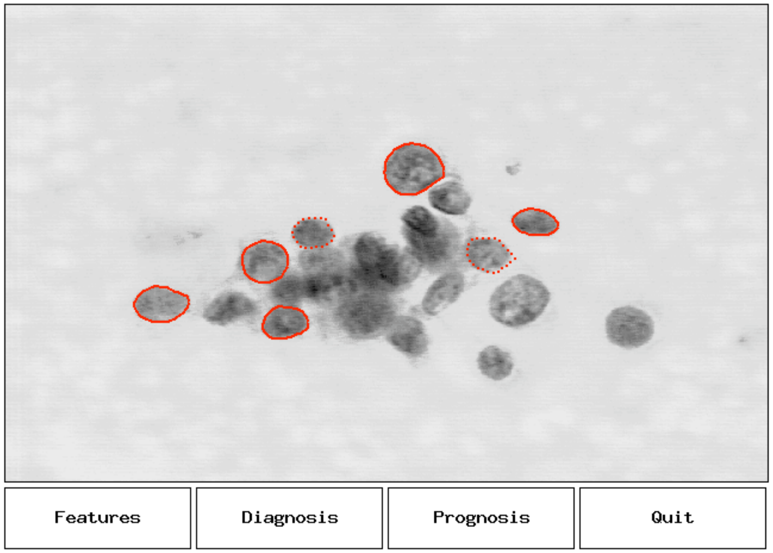
# Passive and active learning

- Traditionally, learning algorithms have been *passive learners*, which take a given batch of data and process it to produce a hypothesis or model

  Data $\rightarrow$ Learner $\rightarrow$ Model
- *Active learners* are instead allowed to query the environment
  - Ask questions
  - Perform experiments
- Open issues: how to query the environment optimally? how to account for the cost of queries?

# Example: A data set

Cell Nuclei of Fine Needle Aspirate



- Cell samples were taken from tumors in breast cancer patients before surgery, and imaged
- Tumors were excised
- Patients were followed to determine whether or not the cancer recurred, and how long until recurrence or disease free

# Data (continued)

- Thirty real-valued variables per tumor.
- Two variables that can be predicted:
  - Outcome (R=recurrence, N=non-recurrence)
  - Time (until recurrence, for R, time healthy, for N).

| tumor size | texture | perimeter | . . . | outcome | time |
|---|---|---|---|---|---|
| 18.02 | 27.6 | 117.5 | | N | 31 |
| 17.99 | 10.38 | 122.8 | | N | 61 |
| 20.29 | 14.34 | 135.1 | | R | 27 |
| . . . | | | | | |

# Terminology

| tumor size | texture | perimeter | . . . | outcome | time |
|---|---|---|---|---|---|
| 18.02 | 27.6 | 117.5 | | N | 31 |
| 17.99 | 10.38 | 122.8 | | N | 61 |
| 20.29 | 14.34 | 135.1 | | R | 27 |
| . . . | | | | | |

- Columns are called *input variables* or *features* or *attributes*
- The outcome and time (which we are trying to predict) are called *output variables* or *targets*
- A row in the table is called *training example* or *instance*
- The whole table is called *(training) data set*.
- The problem of predicting the recurrence is called *(binary) classification*
- The problem of predicting the time is called *regression*

# More formally

| tumor size | texture | perimeter | . . . | outcome | time |
|---|---|---|---|---|---|
| 18.02 | 27.6 | 117.5 | | N | 31 |
| 17.99 | 10.38 | 122.8 | | N | 61 |
| 20.29 | 14.34 | 135.1 | | R | 27 |
| . . . | | | | | |

- A training example $i$ has the form: $\langle x_{i,1}, \dots x_{i,n}, y_i \rangle$ where $n$ is the number of attributes (30 in our case).
- We will use the notation $\mathbf{x}_i$ to denote the column vector with elements $x_{i,1}, \dots x_{i,n}$.
- The training set $D$ consists of $m$ training examples
- We denote the $m \times n$ matrix of attributes by $\mathbf{X}$ and the size-$m$ column vector of outputs from the data set by $\mathbf{y}$.

# Supervised learning problem

- Let $\mathcal{X}$ denote the space of input values
- Let $\mathcal{Y}$ denote the space of output values
- Given a data set $D \subset \mathcal{X} \times \mathcal{Y}$, find a function:

$$h : \mathcal{X} \to \mathcal{Y}$$

  such that $h(\mathbf{x})$ is a *"good predictor"* for the value of $y$.
- $h$ is called a *hypothesis*
- Problems are categorized by the type of output domain
  - If $\mathcal{Y} = \mathbb{R}$, this problem is called *regression*
  - If $\mathcal{Y}$ is a categorical variable (i.e., part of a finite discrete set), the problem is called *classification*

# Steps to solving a supervised learning problem

1. Decide what the input-output pairs are.
2. Decide how to encode inputs and outputs.

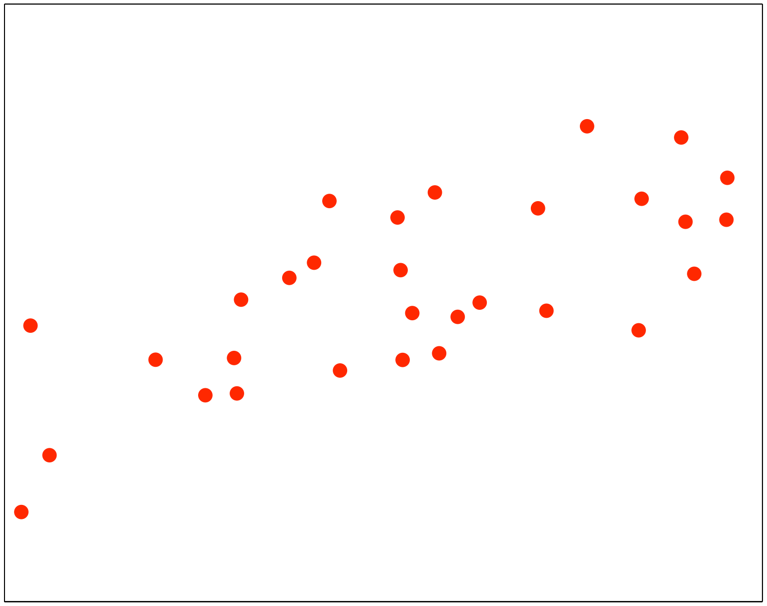
   This defines the input space $\mathcal{X}$, and the output space $\mathcal{Y}$.

   (We will discuss this in detail later)
3. Choose a class of hypotheses/representations $\mathcal{H}$.
4. ...

# Hypothesis class

- All possible hypotheses that we are considering in the search
- Assumptions made on the problem
- You can only be as good as the best hypothesis in the class
- So... Is a larger class always better?
- How can we *control* the size of the class?

# Example: What hypothesis class should we pick?



| $x$ | $y$ |
|---|---|
| 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 |

# Linear hypothesis

- Suppose $y$ was a linear function of $\mathbf{x}$:

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 (+ \cdots)$$

- $w_i$ are called *parameters* or *weights*
- To simplify notation, we can add an attribute $x_0 = 1$ to the other $n$ attributes (also called *bias term* or *intercept term*):

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^{n} w_i x_i = \mathbf{w}^T \mathbf{x}$$

where $\mathbf{w}$ and $\mathbf{x}$ are vectors of size $n + 1$.

*How should we pick* $\mathbf{w}$*?*

# Error minimization!

- Intuitively, $\mathbf{w}$ should make the predictions of $h_{\mathbf{w}}$ close to the true values $y$ on the data we have
- Hence, we will define an *error function* or *cost function* to measure how much our prediction differs from the "true" answer
- We will pick $\mathbf{w}$ such that the error function is minimized

*How should we choose the error function?*

# Least mean squares (LMS)

- Main idea: try to make $h_{\mathbf{w}}(\mathbf{x})$ close to $y$ on the examples in the training set
- We define a *sum-of-squares* error function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2$$

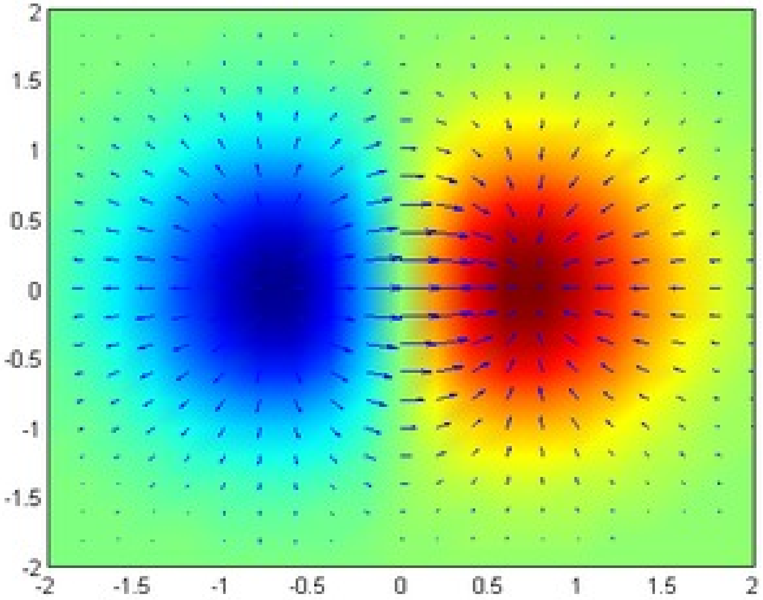
  (the $1/2$ is just for convenience)
- We will choose $\mathbf{w}$ such as to minimize $J(\mathbf{w})$

# Steps to solving a supervised learning problem

1. Decide what the input-output pairs are.
2. Decide how to encode inputs and outputs.

   This defines the input space $\mathcal{X}$, and the output space $\mathcal{Y}$.
3. Choose a class of hypotheses/representations $\mathcal{H}$ .
4. Choose an error function (cost function) to define the best hypothesis
5. Choose an algorithm for searching efficiently through the space of hypotheses.

# Notation reminder: Gradient



- Multivartiate generalization of the derivative.
- Points in the direction of the greatest increase of the function.

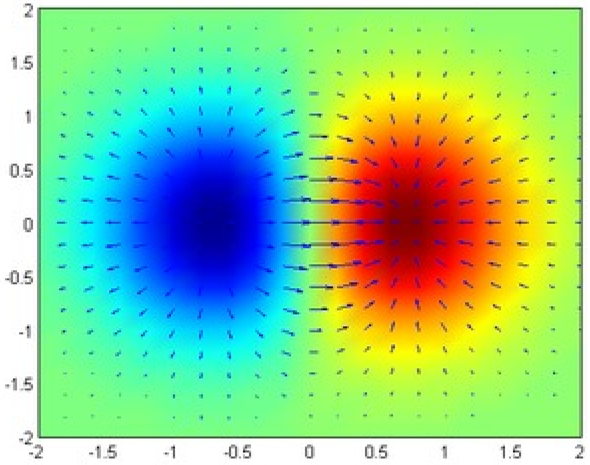
- Consider a function $f(u_1, u_2, \ldots, u_n) : \mathbb{R}^n \mapsto \mathbb{R}$ (for us, this will usually be an error function)
- The *partial derivative* w.r.t. $u_i$ is denoted:

$$\frac{\partial}{\partial u_i} f(u_1, u_2, \ldots, u_n) : \mathbb{R}^n \mapsto \mathbb{R}$$

  The partial derivative is the derivative along the $u_i$ axis, keeping all other variables fixed.
- The *gradient* $\nabla f(u_1, u_2, \ldots, u_n) : \mathbb{R}^n \mapsto \mathbb{R}^n$ is a function which outputs a vector containing the partial derivatives.
  That is:

$$\nabla f = \left\langle \frac{\partial}{\partial u_1} f, \frac{\partial}{\partial u_2} f, \ldots, \frac{\partial}{\partial u_n} f \right\rangle$$

# Properties of the gradient



- The inner product $\langle \nabla f(\mathbf{x}), \mathbf{v} \rangle$ between the gradient of $f$ at $\mathbf{x}$ and any unit vector $\mathbf{v} \in \mathbb{R}^n$ is the *directional derivative* of $f$ in the direction of $\mathbf{v}$ (i.e. the rate at which $f$ changes at $\mathbf{x}$ in the direction $\mathbf{v}$).

$\rightarrow$ $\nabla f(\mathbf{x})$ points towards the direction of greatest increase of $f$ at $\mathbf{x}$.

$\rightarrow$ Points such that $\nabla f(\mathbf{x}) = \mathbf{0}$ are called stationary points.

$\rightarrow$ The gradient $\nabla f(\mathbf{x})$ is orthogonal to the contour line passing through $\mathbf{x}$.

# 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 linear system with $(n+1)$ equations and $(n+1)$ unknowns.

# The solution

- Concise form of the error function: $J(\mathbf{w}) = \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$.
- Recalling some multivariate calculus:

$$\begin{aligned}
\nabla_{\mathbf{w}} J &= \nabla_{\mathbf{w}} \frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= \frac{1}{2}\nabla_{\mathbf{w}}(\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{y}^T\mathbf{X}\mathbf{w} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{y}^T\mathbf{y}) \\
&= \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y}
\end{aligned}$$

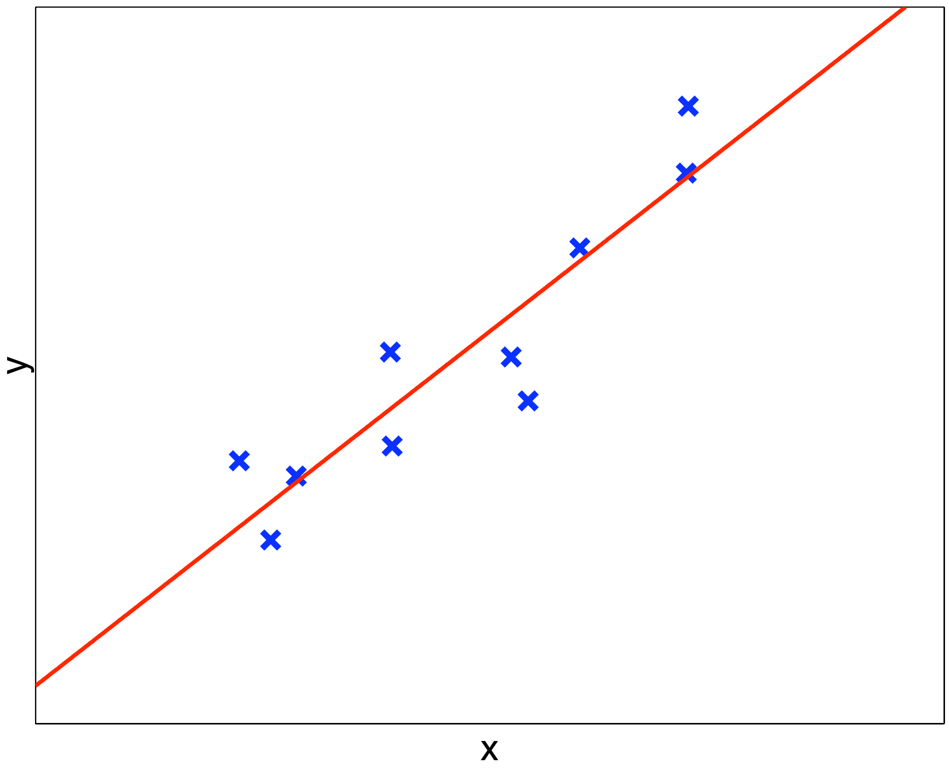
- Setting gradient equal to zero:

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

- The inverse exists if the columns of $\mathbf{X}$ are linearly independent.

# Example: Data and best linear hypothesis

$$y = 1.60x + 1.05$$

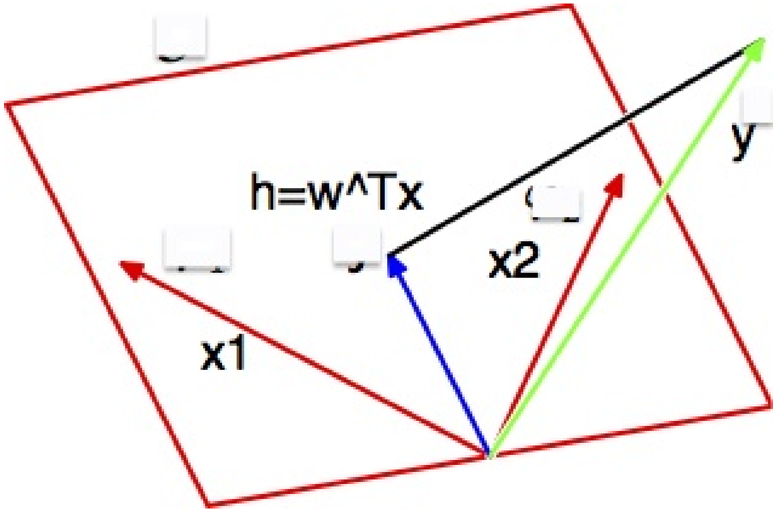


# Linear regression summary

- The optimal solution (minimizing sum-squared-error) can be computed in polynomial time in the size of the data set.
- The solution is $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$, where $\mathbf{X}$ is the data matrix augmented with a column of ones, and $\mathbf{y}$ is the column vector of target outputs.
- A very rare case in which an analytical, exact solution is possible

# Coming back to mean-squared error function...

- Good intuitive feel (small errors are ignored, large errors are penalized)
- Nice math (closed-form solution, unique global optimum)
- Geometric interpretation



- Any other interpretation?

# A probabilistic assumption

- Assume $y_i$ is a noisy target value, generated from a hypothesis $h_{\mathbf{w}}(\mathbf{x})$
- More specifically, assume that there exists $\mathbf{w}$ such that:

$$y_i = h_{\mathbf{w}}(\mathbf{x}_i) + \epsilon_i$$

  where $\epsilon_i$ is random variable (noise) drawn independently for each $\mathbf{x}_i$ according to some Gaussian (normal) distribution with mean zero and variance $\sigma^2$.

- How should we choose the parameter vector $\mathbf{w}$?

# Maximum Likelihood Estimation

- Let $h$ be a hypothesis and $D$ be the set of training data.
- The *likelihood of the data* given the hypothesis $h$ is the probability that the data $D$ has been generated by hypothesis $h$:

$$\mathcal{L}(h) = P(D \mid h).$$

- Maximum Likelihood (ML) hypothesis. We can try to find the hypothesis that maximizes the likelihood of the data:

$$h_{ML} = \arg\max_{h \in \mathcal{H}} \mathcal{L}(h).$$

- Standard assumption: the training examples are *independently identically distributed (i.i.d.)*

- This allows us to simplify the likelihood:

$$\mathcal{L}(h) = P(D \mid h) = \prod_{i=1}^{m} P(\mathbf{x_i}, y_i \mid h) = \prod_{i=1}^{m} P(y_i \mid \mathbf{x}_i, h) P(\mathbf{x}_i)$$

(for discrete distribution $P$ is the probability, for continuous distribution $P$ would be the density).

# The $\log$ trick

- We want to maximize:

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

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

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

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

- The second sum depends on $D$, but not on $h$, so it can be ignored in the search for a good hypothesis

# Maximum likelihood for regression

- Recall our assumption:

$$y_i = h_{\mathbf{w}}(\mathbf{x}_i) + \epsilon_i,$$

  where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.

- What is the distribution of $y_i$ given $\mathbf{x}_i$?

# Maximum likelihood for regression

- Recall our assumption:

$$y_i = h_{\mathbf{w}}(\mathbf{x}_i) + \epsilon_i,$$

  where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.

- Hence, each $y_i$ (given $\mathbf{x}_i$) is drawn from the Gaussian distribution $\mathcal{N}(h_{\mathbf{w}}(\mathbf{x}_i), \sigma^2)$.
- Since the $y_i$'s are independent, we have

$$\mathcal{L}(\mathbf{w}) = \prod_{i=1}^{m} p(y_i|\mathbf{x}_i, \mathbf{w}, \sigma) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{y_i - h_{\mathbf{w}}(\mathbf{x}_i)}{\sigma}\right)^2}$$

# Applying the $\log$ trick

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

Maximizing the right hand side is the same as minimizing:

$$\sum_{i=1}^{m} \frac{1}{2}\frac{(y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2}{\sigma^2}$$

This is our old friend, the sum-squared-error function! (the constants that are independent of $h$ can again be ignored)

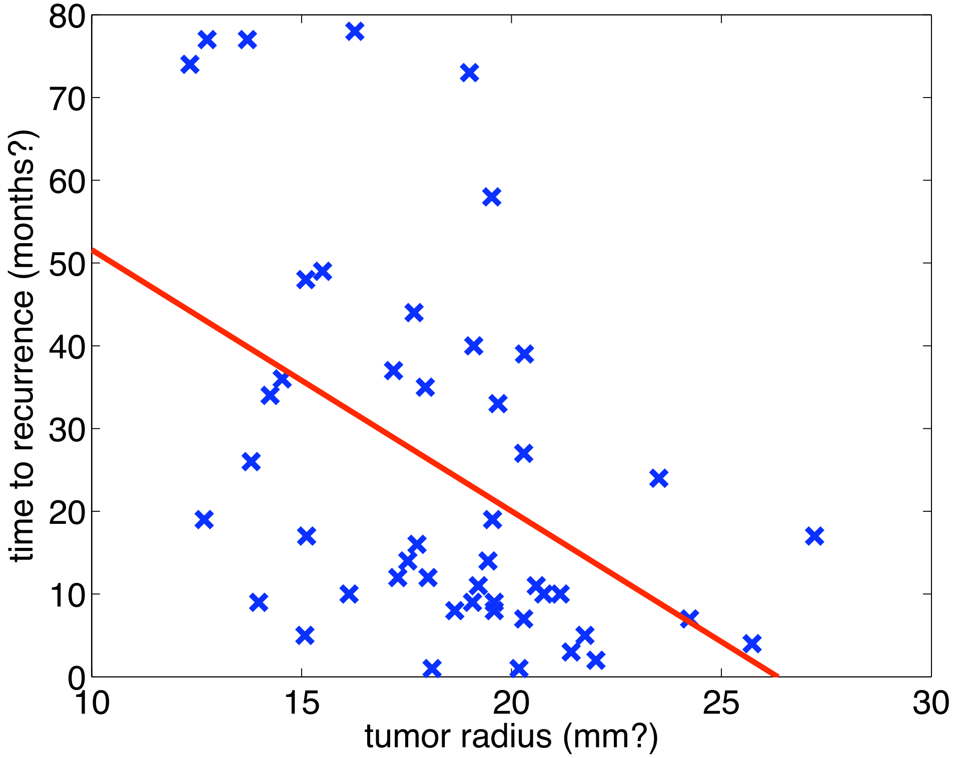
# Maximum likelihood hypothesis for least-squares estimators

- Under the assumption that the training examples are i.i.d. and that we have *Gaussian target noise*, the maximum likelihood parameters $\mathbf{w}$ are those minimizing the sum squared error:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x_i}))^2$$

- This makes explicit the hypothesis behind minimizing the sum-squared error
- If the noise is not normally distributed, maximizing the likelihood will not be the same as minimizing the sum-squared error
- In practice, different loss functions are used depending on the noise assumption

# Predicting recurrence time based on tumor size



# Is linear regression enough?

- Linear regression is too simple for most realistic problems

  But it should be the first thing you try for real-valued outputs!
- Problems can also occur when $\mathbf{X}^T\mathbf{X}$ is not invertible (though this can be dealt with, more on that later).
- Two possible solutions:
  1. Transform the data
     - Add cross-terms, higher-order terms
     - More generally, apply a transformation of the inputs from $\mathcal{X}$ to some other space $\mathcal{X}'$, then do linear regression in the transformed space
  2. Use a different hypothesis class (e.g. non-linear functions)
- Today we focus on the first approach

# Polynomial fits

- Suppose we want to fit a higher-degree polynomial to the data. (E.g., $y = w_2x^2 + w_1x^1 + w_0$.)
- Suppose for now that there is a single input variable per training sample.
- How do we do it?

# Answer: Polynomial regression

- Given data: $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$.
- Suppose we want a degree-$d$ polynomial fit for the data

$$\mathbf{X} = [x_1, \cdots, x_m]^\top, \quad \mathbf{y} = [y_1, \cdots, y_m]^\top$$

- Let $\mathbf{y}$ be as before and let

$$\mathbf{\Phi} = \left[ \begin{array}{ccccc} x_1^d & \ldots & x_1^2 & x_1 & 1 \\ x_2^d & \ldots & x_2^2 & x_2 & 1 \\ \vdots & & \vdots & \vdots & \vdots \\ x_m^d & \ldots & x_m^2 & x_m & 1 \end{array} \right]$$

- Solve the linear regression $\mathbf{\Phi w} \approx \mathbf{y}$.

# Example of quadratic regression: Data matrices

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

# $\mathbf{\Phi}^\top\mathbf{\Phi}$

$$\mathbf{\Phi}^\top\mathbf{\Phi} =$$

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

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

# $\mathbf{\Phi}^T\mathbf{y}$

$$\mathbf{\Phi}^T\mathbf{y} =$$

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

$$= \left[\begin{array}{c} 3.60 \\ 6.49 \\ 8.34 \end{array}\right]$$

# Solving for $\mathbf{w}$

$$\begin{aligned}
\mathbf{w} &= (\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{y} \\
&= \begin{bmatrix} 4.11 & -1.64 & 4.95 \\ -1.64 & 4.95 & -1.39 \\ 4.95 & -1.39 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 3.60 \\ 6.49 \\ 8.34 \end{bmatrix} = \begin{bmatrix} 0.68 \\ 1.74 \\ 0.73 \end{bmatrix}
\end{aligned}$$

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

# Linear function approximation in general

- Given a set of examples $(\mathbf{x}_i, y_i)_{i=1\ldots m}$, we fit a hypothesis

  $$h_{\mathbf{w}}(\mathbf{x}) = \sum_{k=0}^{K-1} w_k \phi_k(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

  where $\phi_k$ are called basis functions
- The best $\mathbf{w}$ is considered the one which minimizes the sum-squared error over the training data:

  $$\sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2$$

- We can find the best $\mathbf{w}$ in closed form:

  $$\mathbf{w} = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{y}$$

  or by other methods (e.g. gradient descent - as will be seen later)

# Linear models in general

- By linear models, we mean that the hypothesis function $h_{\mathbf{w}}(\mathbf{x})$ is a *linear function of the parameters* $\mathbf{w}$
- This *does not mean the* $h_{\mathbf{w}}(\mathbf{x})$ *is a linear function of the input vector* $\mathbf{x}$ (e.g., polynomial regression)
- In general

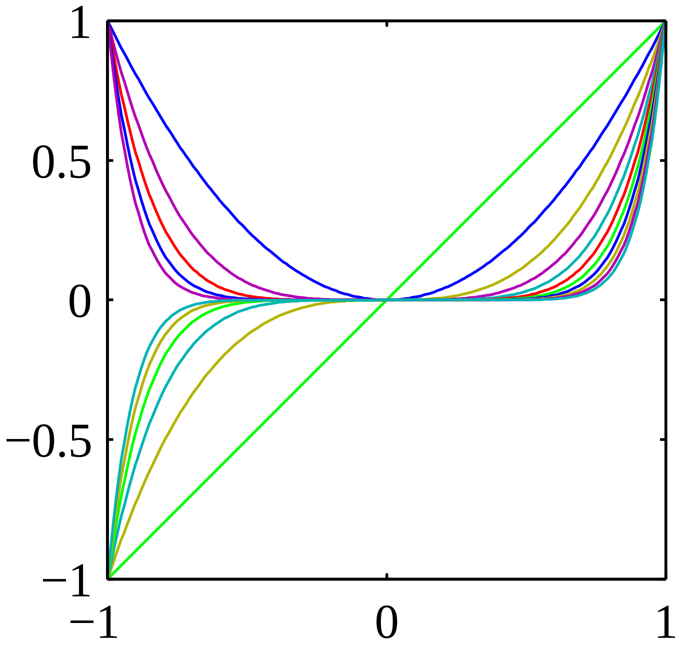
$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{k=0}^{K-1} w_k \phi_k(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

  where $\phi_k$ are called *basis functions*
- Usually, we will assume that $\phi_0(\mathbf{x}) = 1, \forall \mathbf{x}$, to create a bias term
- The hypothesis can alternatively be written as:

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{\Phi} \mathbf{w}$$

  where $\mathbf{\Phi} \in \mathbb{R}^{m \times l}$ is a matrix with one row per instance: $\mathbf{\Phi}_{j,:} = \phi(\mathbf{x}_j)^\top$.
- Basis functions are *fixed*

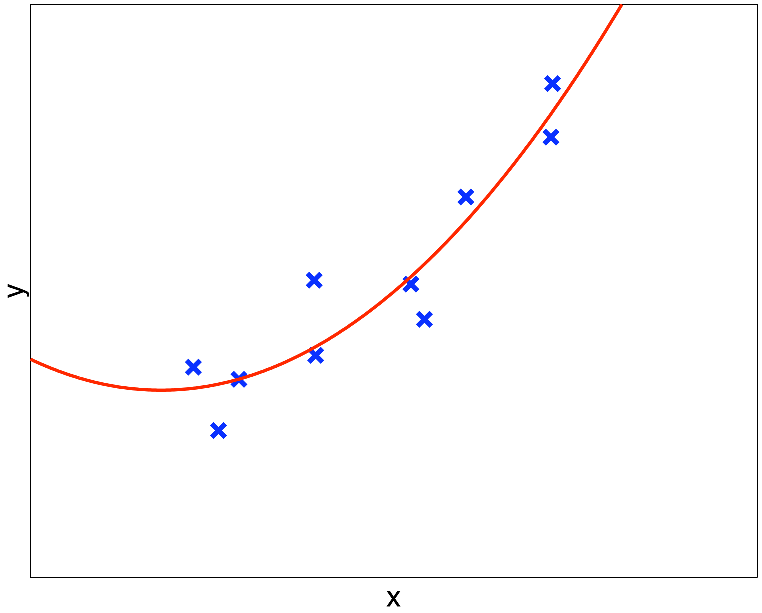
# Example basis functions: Polynomials



$$\phi_k(x) = x^k$$

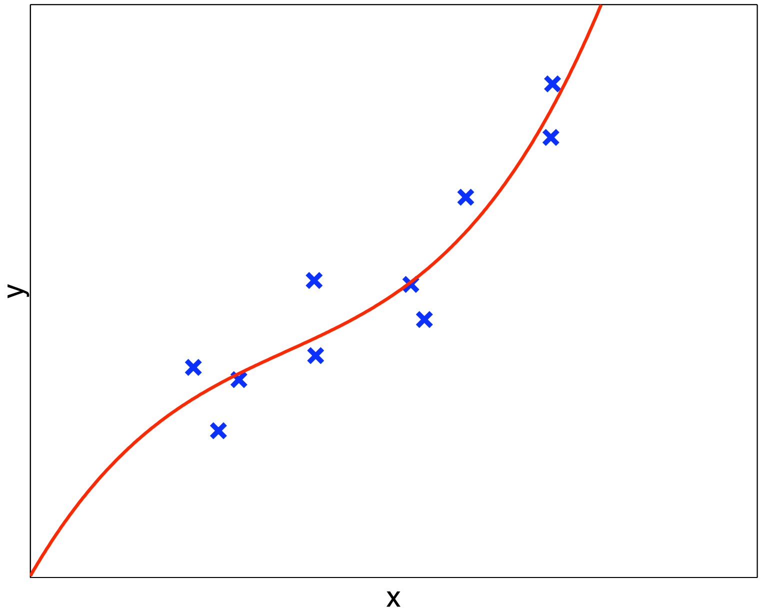
"Global" functions: a small change in $x$ may cause large change in the output of many basis functions

# Order-2 fit



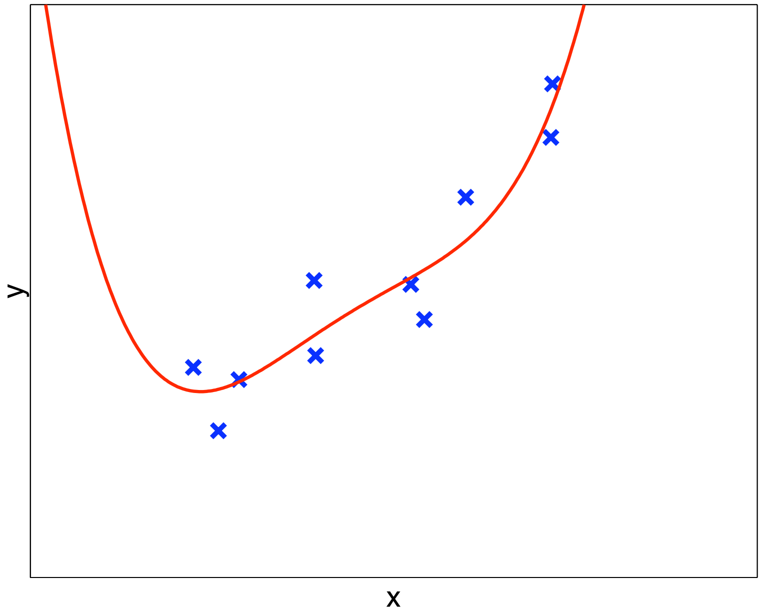
Is this a better fit to the data?

# Order-3 fit



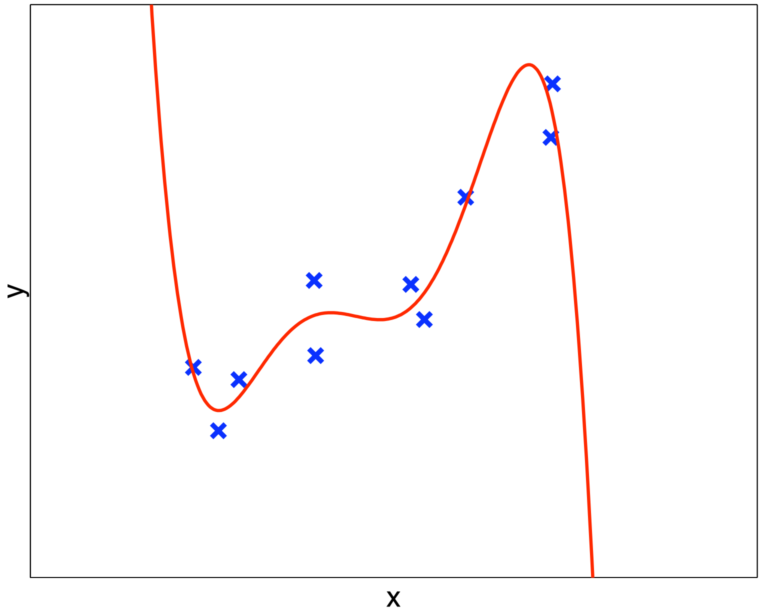
Is this a better fit to the data?

# Order-4 fit



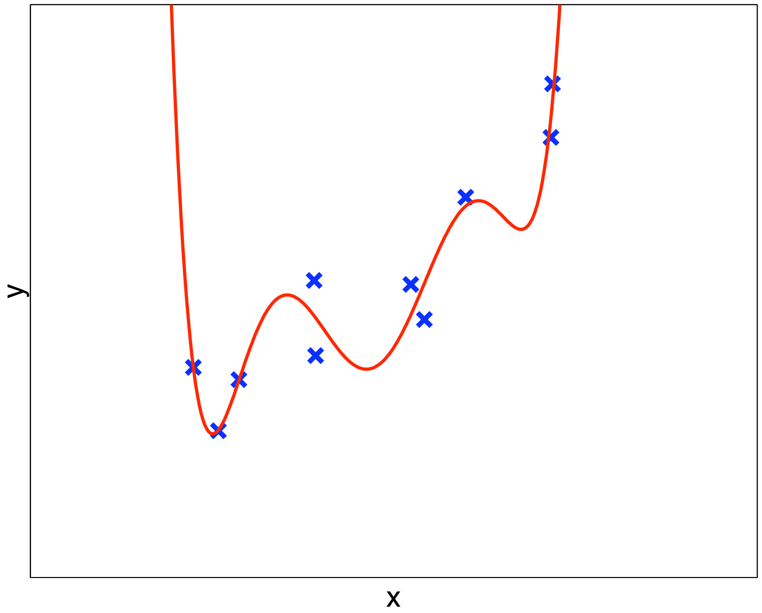
Is this a better fit to the data?

# Order-5 fit



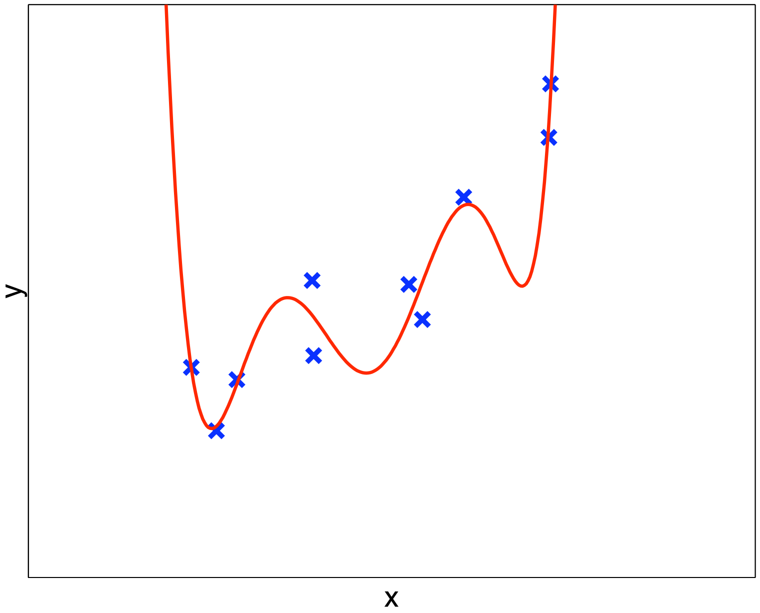
Is this a better fit to the data?

# Order-6 fit



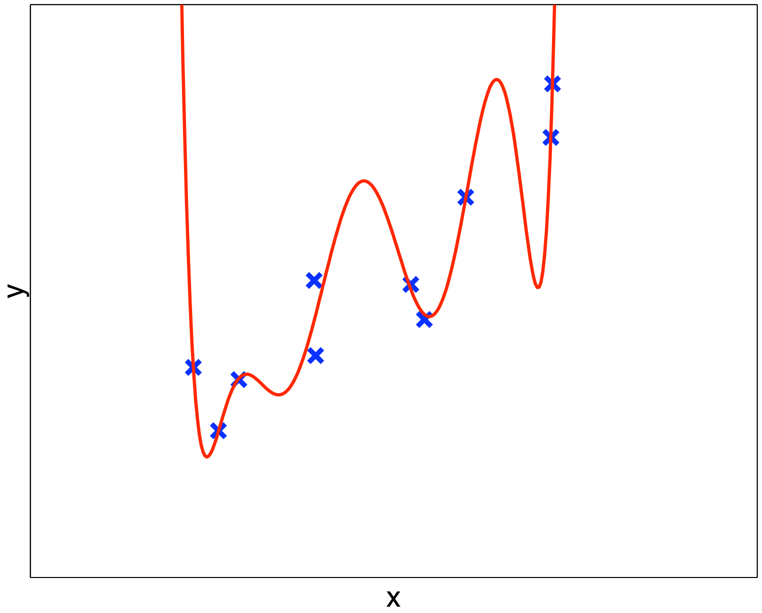
Is this a better fit to the data?

# Order-7 fit



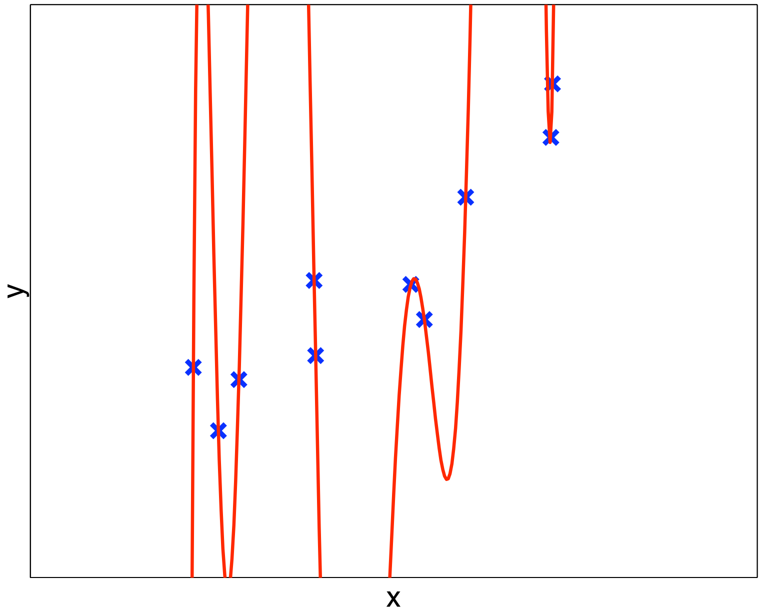
Is this a better fit to the data?

# Order-8 fit



Is this a better fit to the data?

# Order-9 fit



Is this a better fit to the data?

# 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.
- Next time: defining overfitting formally, and finding ways to avoid it