COMP 6321
Machine Learning

Instructor: Adam Krzyżak
Email: krzyzak@cs.concordia.ca

Teaching assistant: Mina Yousefi

Email: mi_yousu@encs.concordia.ca

Class web page:
http://users.encs.concordia.ca/~c63214/index.html

Outline

- Administrative issues
- What is machine learning?
- Types of machine learning
- Supervised learning
- Linear and polynomial regression
- Overfitting
- Cross-validation

Administrative issues

- Class materials:
  - Recommended textbooks:
    - Christopher M. Bishop, "Pattern Recognition and Machine Learning", Springer, 2006.
    - Kevin P. Murphy, Machine Learning. A Probabilistic Perspective, MIT Press, 2012
    - Richard O. Duda, Peter E. Hart and David G. Stork, Pattern Classification, Wiley, Second Edition, 2001.
  - Additional textbooks (see syllabus)
  - Additional readings: distributed in class and/or posted on web
  - Class notes: posted on the web page
- Prerequisites:
  - Knowledge of a programming language
  - Some knowledge of probability/statistics and linear algebra; general facility with math

- Some AI background is recommended *but not required*
- Matlab will be used in assignments but prior knowledge is *not required*

Evaluation

- Up to five homework assignments (50%)
- Midterm examination (20%)
- Final project (30%)

What is learning?

- H. Simon: Any process by which a system improves its performance
- M. Minsky: Learning is making useful changes in our minds
- R. Michalsky: Learning is constructing or modifying representations of what is being experienced
- L. Valiant: Learning is the process of knowledge acquisition in the absence of explicit programming

Why study machine learning?

- 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
- Discover knowledge and patterns in highly dimensional, complex data
  - Sky surveys
  - Sequence analysis in bioinformatics
  - Social network analysis
  - Ecosystem analysis

Why study machine learning?

- Solving tasks that require a system to be adaptive, e.g.
  - Speech and handwriting recognition
  - "Intelligent" user interfaces
- 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)
- 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, COLT), major journals (e.g., Journal of Machine Learning Research (free online), Machine Learning, Neural Computation, IEEE Transactions on Neural Networks and Learning Systems)
- The time is right to study in the field!
  - Lots of recent progress in algorithms and theory

- Growing flood of data to be analyzed
- Computational power is available
- Growing demand for industrial applications
- Important for web applications (Google, Facebook, Microsoft)

Related disciplines

- Artificial intelligence
- Probability theory and statistics
- Computational complexity theory
- Control theory
- Information theory
- Philosophy
- Psychology and neurobiology

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, face recognition, vision
- 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

Supervised learning

- Training experience: a set of *labeled examples* of the form

$$\langle x_1\, x_2\, \ldots x_n, y \rangle,$$

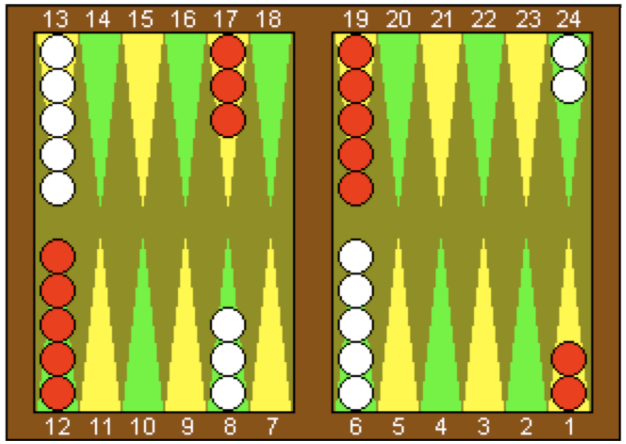
  where x_j are values for *input variables* and y is the *output*
- This implies the existence of a "teacher" who knows the right answers
- What to learn: A *function* $f : X_1 \times X_2 \times \cdots \times X_n \to Y$, which maps the input variables into the output domain
- Goal: minimize the error (loss function) on the training examples

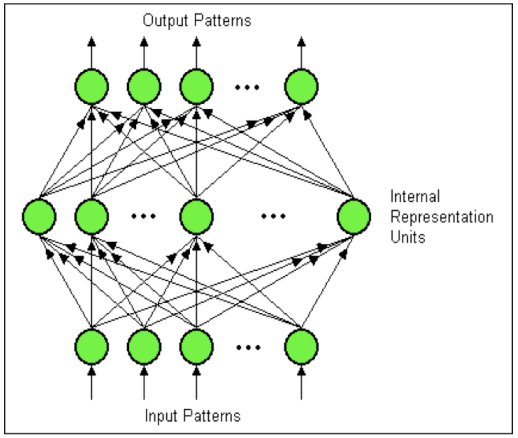
Example: Face detection and recognition

Reinforcement learning

- Training experience: interaction with an environment; the agent receives a numerical reward signal
- E.g., a trading agent in a market; the reward signal is the profit
- What to learn: a way of behaving that is very rewarding in the long run
- Goal: estimate and maximize the long-term cumulative reward

Example: TD-Gammon (Tesauro)



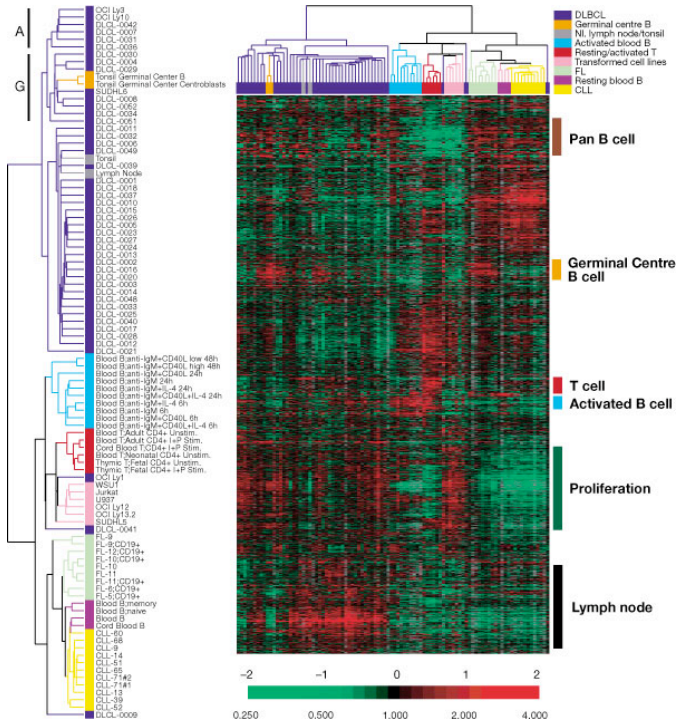


- A computer backgammon program developed in 1992 by Gerald Tesauro of IBM
- Used neural network and evaluation function. Learning from self-play, using TD-learning
- Became the best player in the world
- Discovered new ways of opening not used by people before

Unsupervised learning

- Training experience: unlabeled data
- What to learn: interesting associations in the data
- E.g., image segmentation, clustering
- Often there is no single correct answer

Example: Oncology (Alizadeh et al.)



- Activity levels of all ($\approx$ 25,000) genes were measured in lymphoma patients
- Cluster analysis determined three different subtypes (where only two were known before), having different clinical outcomes

Passive and active learning

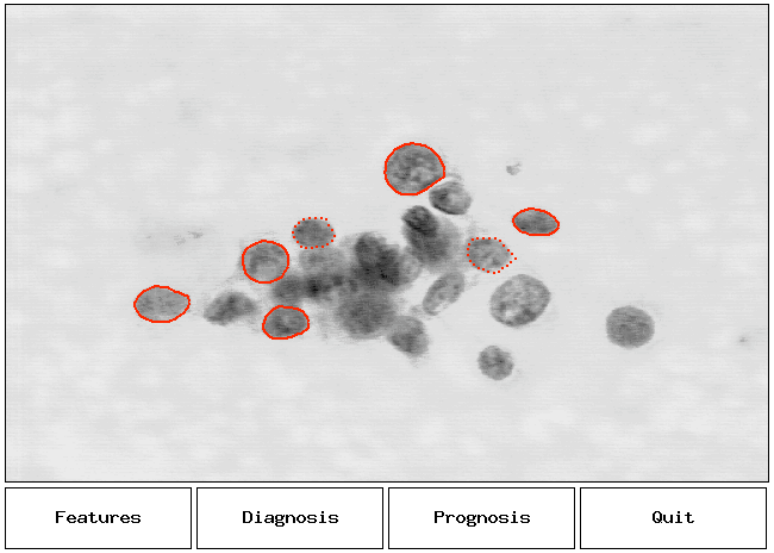
- Traditionally, learning algorithms have been *passive learners*, which take a given batch of data and process it to produce a hypothesis or model

  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 for supervised learning

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

Wisconsin 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*.

Prediction problems

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
...					

- 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: $\langle x_{i,1}, \ldots x_{i,n}, y_i \rangle$ 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 $x_{i,1}, \ldots x_{i,n}$.
- The training set D consists of m training examples
- We denote the $m \times n$ matrix of attributes by X and the size-m column vector of outputs from the data set by Y.
- Let $\mathcal{X}$ denote the space of input values
- Let $\mathcal{Y}$ denote the space of output values

Supervised learning problem

- Given a data set $D \subset \mathcal{X} \times \mathcal{Y}$, find a function:

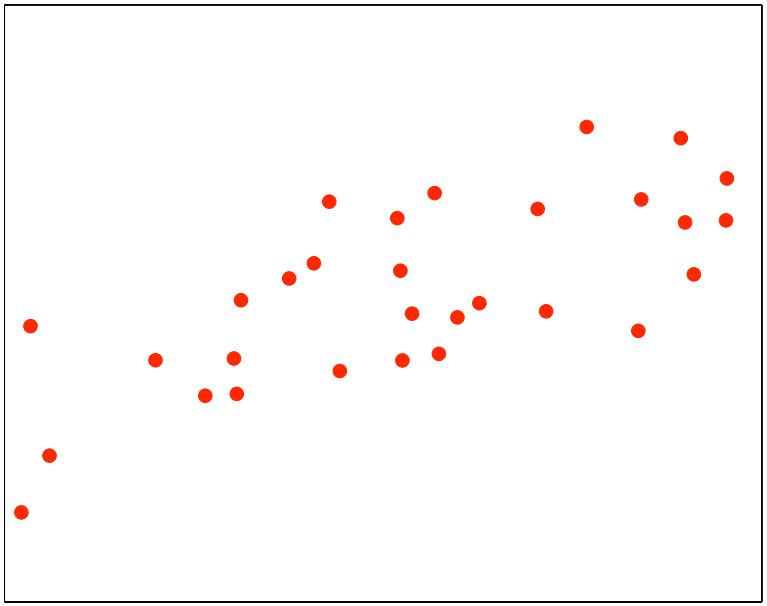
$$h : \mathcal{X} \to \mathcal{Y}$$

  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 finite discrete set, the problem is called *classification*
  - If $\mathcal{Y}$ has 2 elements, the problem is called *binary classification* or *concept learning*

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. ...

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 $J(\mathbf{w})$
- One way to do it: compute $\mathbf{w}$ such that:

$$\frac{\partial}{\partial w_j}J(\mathbf{w}) = 0, \ \forall j = 0 \ldots n$$

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

- 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:

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

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

- Recalling some multivariate calculus:

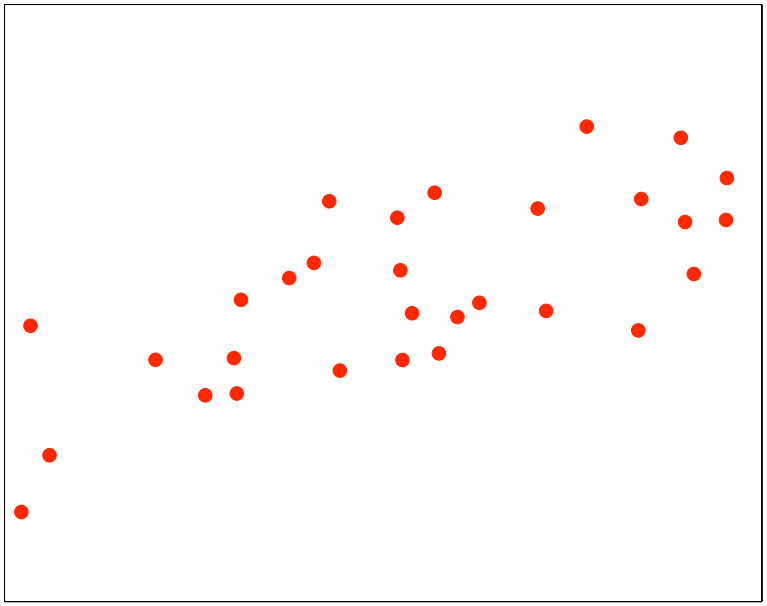
$$\begin{aligned}
\nabla_{\mathbf{w}} J &= \nabla_{\mathbf{w}} (X\mathbf{w} - Y)^T (X\mathbf{w} - Y) \\
&= \nabla_{\mathbf{w}} (\mathbf{w}^T X^T X\mathbf{w} - Y^T X\mathbf{w} - \mathbf{w}^T X^T Y + Y^T Y) \\
&= 2X^T X\mathbf{w} - 2X^T Y
\end{aligned}$$

- 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.

Example of linear regression



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

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

$X^T X$

$$X^T X =$$

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

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

$X^T Y$

$$X^T Y =$$

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

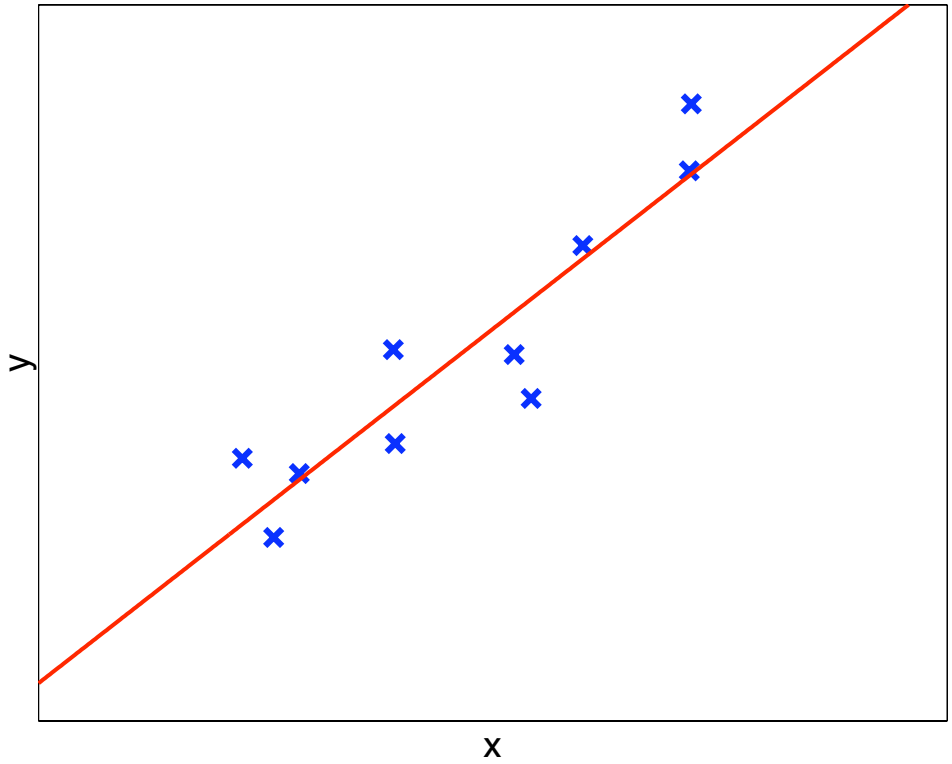
$$= \begin{bmatrix} 6.49 \\ 8.34 \end{bmatrix}$$

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 $y = 1.60x + 1.05$

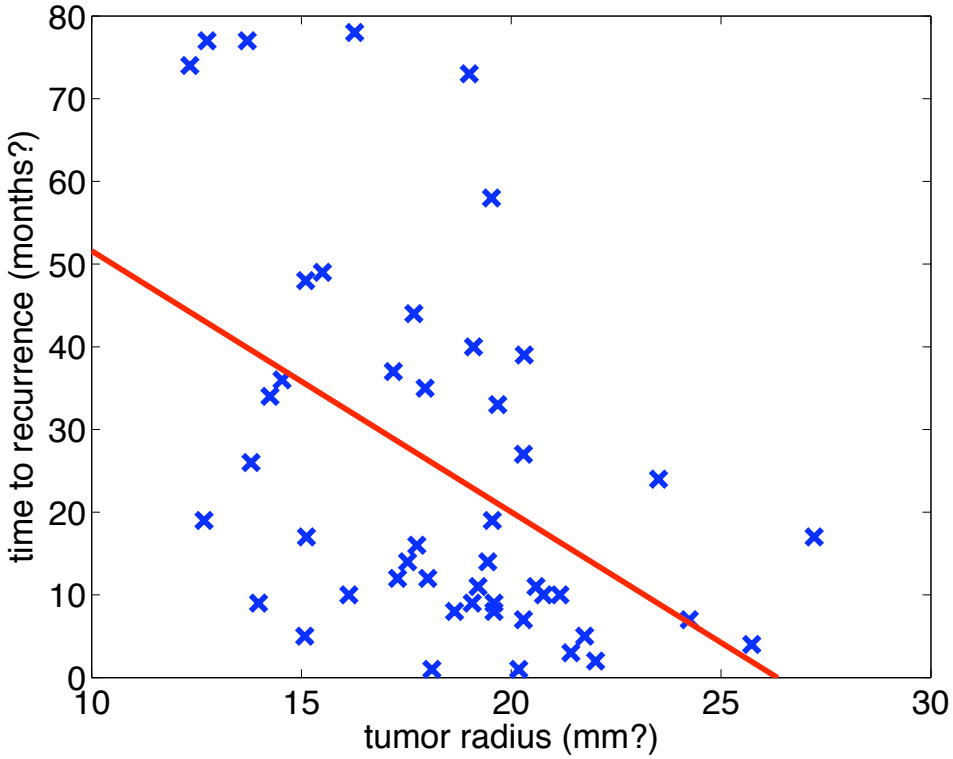


Linear regression summary

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

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!
- Problems can also occur is $X^T X$ is not invertible.
- 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
- 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.
- Let Y be as before and let

$$X = \begin{bmatrix} x_1^d & \ldots & x_1^2 & x_1 & 1 \\ x_2^d & \ldots & x_2^2 & x_2 & 1 \\ \vdots & & \vdots & \vdots & \vdots \\ x_m^d & \ldots & x_m^2 & x_m & 1 \end{bmatrix}$$

- Solve the linear regression $X\mathbf{w} \approx Y$.

Example of quadratic regression: Data matrices

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

$X^T X$

$$X^T X =$$

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

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

$X^T Y$

$$X^T Y =$$

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

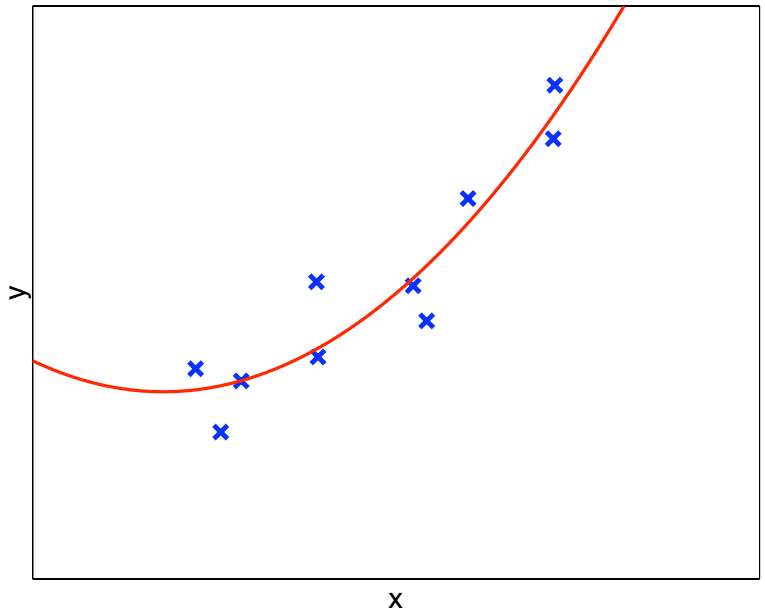
$$= \begin{bmatrix} 3.60 \\ 6.49 \\ 8.34 \end{bmatrix}$$

Solving for $\mathbf{w}$

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

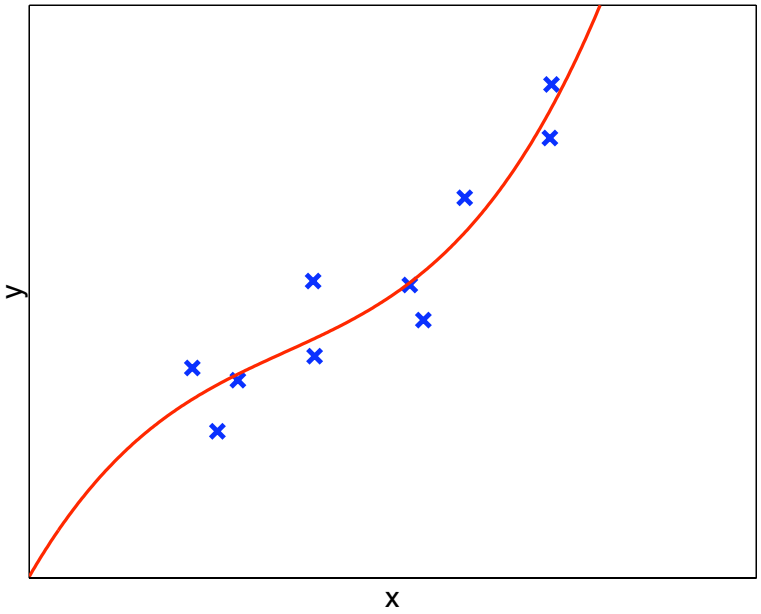
So the best order-2 polynomial is $y = 0.68x^2 + 1.74x + 0.73$.

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



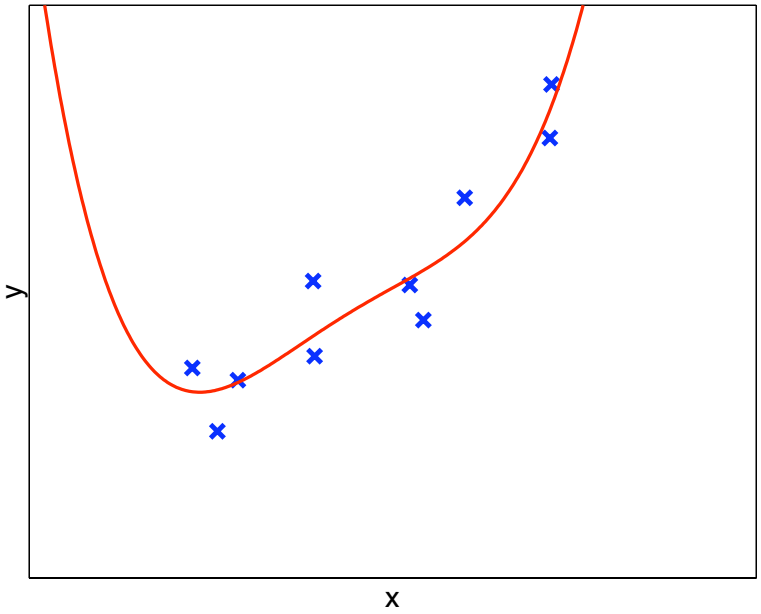
Is this a better fit to the data?

Order-3 fit



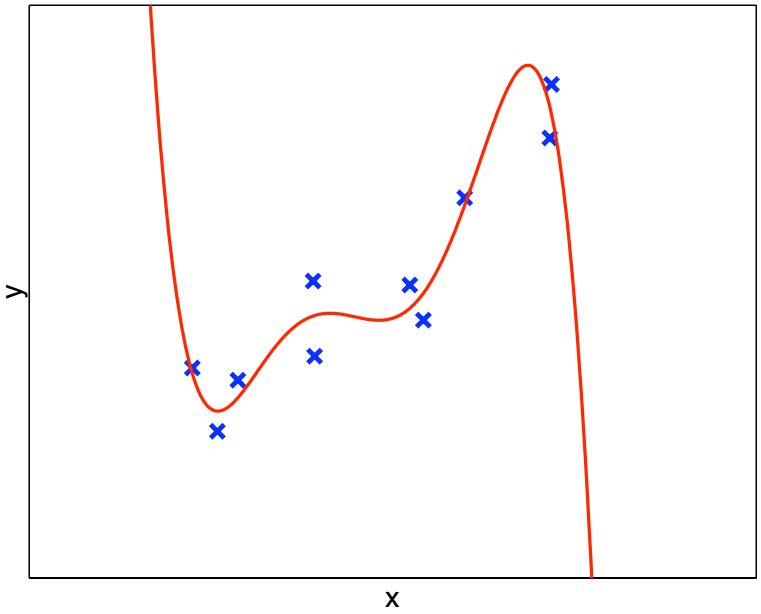
Is this a better fit to the data?

Order-4 fit



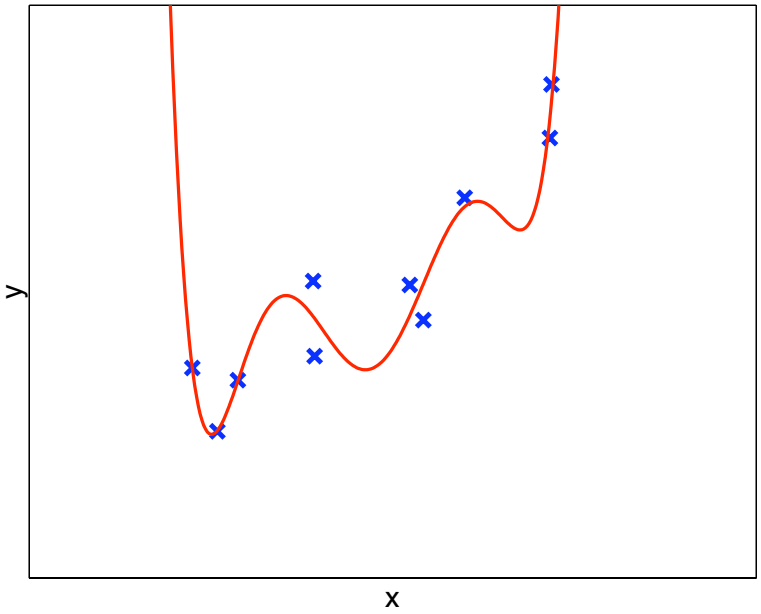
Is this a better fit to the data?

Order-5 fit



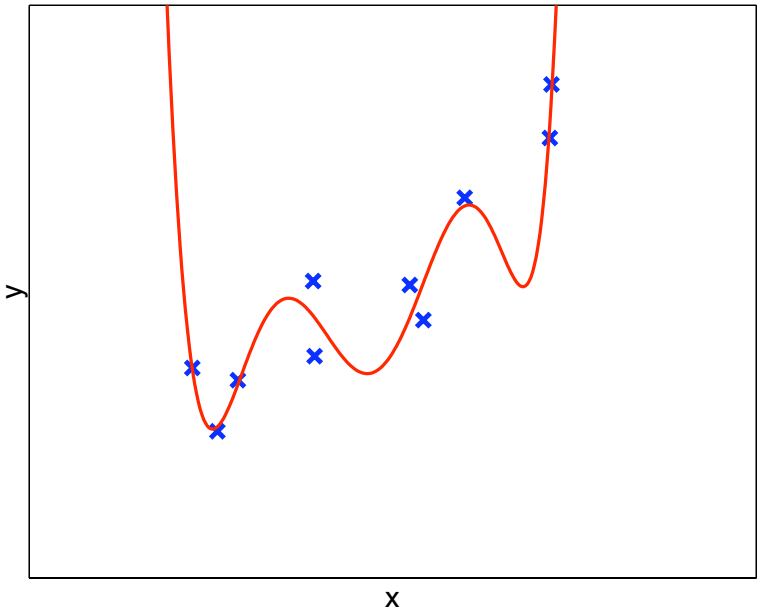
Is this a better fit to the data?

Order-6 fit



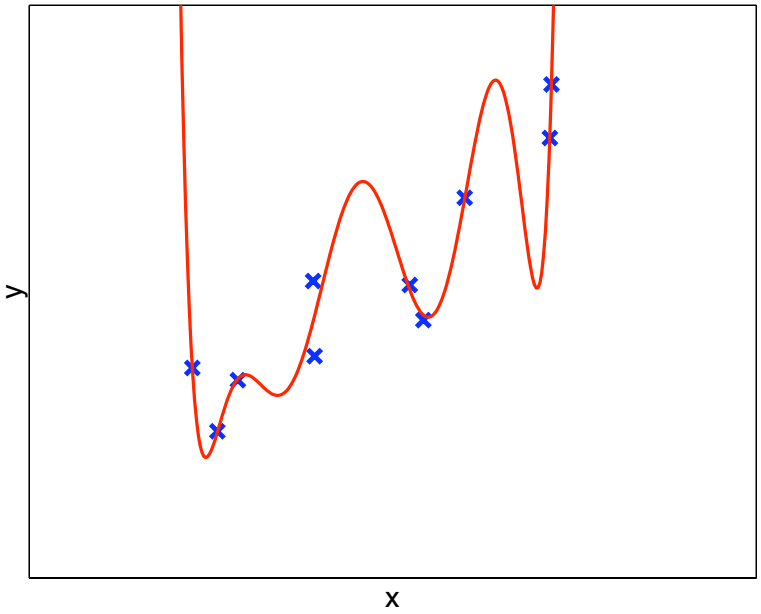
Is this a better fit to the data?

Order-7 fit



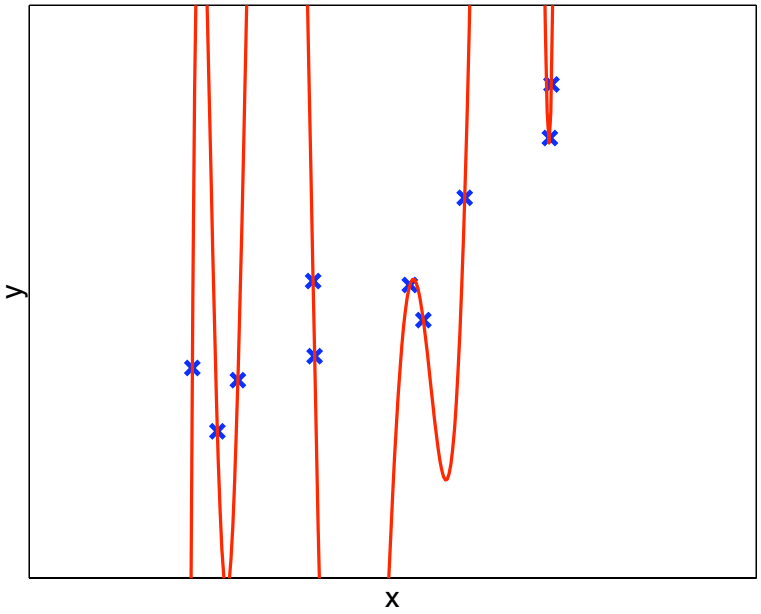
Is this a better fit to the data?

Order-8 fit



Is this a better fit to the data?

Order-9 fit



Is this a better fit to the data?

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

- Assume that the data is drawn from some fixed, unknown probability distribution

  We will discuss later details about the drawing process and the types of distributions
- Every hypothesis has a "true" error $J^*(h)$, which is the expected error when data is drawn from the distribution.
- 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 m times:
  - 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 instances to find $\mathbf{w}$
  - Measure the error in predicting the label on the instance left out, for the $\mathbf{w}$ parameter vector
  - This is an *unbiased estimate of the true prediction error*
- Choose the d with lowest average prediction error (averaged over all the instances)

Estimating true error for $d = 1$

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

Iter	D_{train}	D_{valid}	$\text{Error}_{\text{train}}$	$\text{Error}_{\text{valid}}$
1	$D - \{(0.86, 2.49)\}$	$(0.86, 2.49)$	0.4928	0.0044
2	$D - \{(0.09, 0.83)\}$	$(0.09, 0.83)$	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

Leave-one-out 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

- Typical overfitting behavior: as d increases, the training error decreases, but the testing error decreases, then starts increasing again
- Optimal choice: $d = 2$. Overfitting for $d > 2$
- Why are $d = 8$ and $d = 9$ the same?

Cross-validation

- 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 sets *must be disjoint*!
- The process is repeated several times, and the results are averaged to provide error estimates.

k-fold cross-validation

- Divide the instances into k disjoint partitions or folds
- Loop through the partitions:
  - Partition i is for testing (i.e., estimating the performance of the algorithm after learning is done)
  - One of the other partitions is for validation (e.g., choosing the hypothesis class or the parameters of the learning algorithm)
  - The rest are used for training (e.g., choosing the specific hypothesis within the class)
- *Report average error on the testing partitions*
- Reporting error on any other data gives you an *optimistic estimate* of performance
- You can also compute standard deviation based on the testing errors on the different folds
- Magic number: $k = 10$
- We will discuss later how to end up producing just one hypothesis

Polynomial regression (more generally)

- Suppose you have several input variables
- Instead of just the higher-order terms, we can add them and all the cross-product terms, up to degree d
- What happens with the size of $\mathbf{w}$?
- What do you expect in terms of overfitting?

Summary

- We can fit linear and polynomial functions in polynomial time by solving $\mathbf{w} = (X^T X)^{-1} X^T Y$.
- We can use cross-validation to choose the best order of polynomial to fit our data.
- Because the number of parameters explodes exponentially with the degree of the polynomial, we will often use powers of individual input variables but no cross terms, or only select cross-terms (based on domain knowledge).
- *Always use cross-validation, and report results on testing data (completely untouched in the training process)*