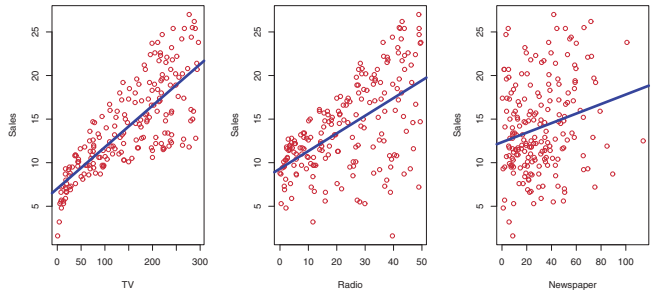
Tufts

Class #02: Linear & Polynomial Regression; Error and Over-fitting

Machine Learning (COMP 135)

1

Practical Use of Linear Regression

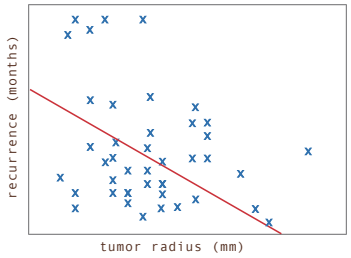


Ad sales vs. media expenditure (1000's of units). From: James et al., *Intro. to Statistical Learning* (Springer, 2017)

- A linear model can often radically simplify a data-set, isolating a relatively straightforward relationship between data-features and outcomes

2

Accuracy of the Hypothesis Function



- Although we can generally find the best set of weights efficiently, the exact form of the equation, in terms of the ***degree*** of the polynomial used in that equation, can limit our accuracy
- ***Example***: if we try to predict time to tumor recurrence based on a simple linear function of its radius, this is likely to be very inaccurate

3

Higher Order Polynomial Regression

- Since not every data-set is best represented as a simple linear function, we will in general want to explore ***higher-order*** hypothesis functions
- We can still keep these functions quasi-linear, in terms of a sum of weights over terms, but we will allow those terms to take more complex polynomial forms, like:

$$h(x) \longleftarrow y = w_0 + w_1 x + w_2 x^2$$

4

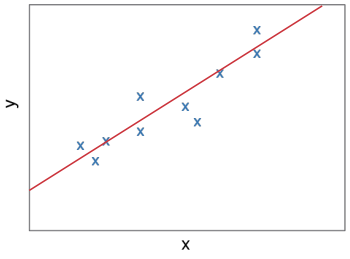
Higher Order Polynomial Regression

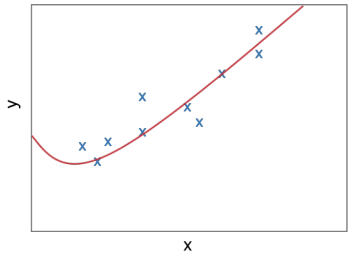
$$h(x) \longleftarrow y = w_0 + w_1 x + w_2 x^2$$

- Note: the hypothesis function here is ***still*** linear, in terms of a sum of coefficients, each multiplied by a single feature
  - The same algorithms can find the coefficients that minimize error, just as before
- What is different, however, are the ***features*** themselves
  - A feature transformation is a common ML technique
  - In order to best solve a problem, we generally ***don't care*** what features we use
  - We will often experiment with modifying features to get better results from existing algorithms

5

Higher-Order Regression Solutions



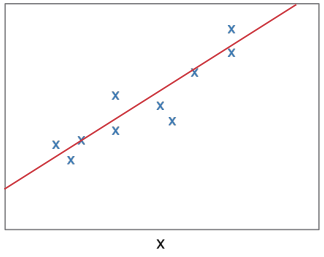


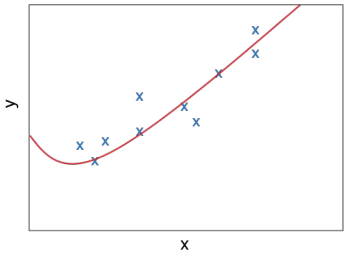
$h(x) \longleftarrow y = 1.05 + 1.60x$ $h(x) \longleftarrow y = 0.73 + 1.74x + 0.68x^2$

- With an order-2 function, we can fit our data somewhat better than with the original, order-1 version

6

Higher-Order Regression Solutions





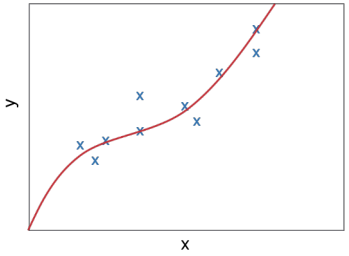
$h(x) \longleftarrow y = 1.05 + 1.60x$ $h(x) \longleftarrow y = 0.73 + 1.74x + 0.68x^2$

- It is important to note that the "curves" we get are still linear
  - These are the result of projecting a linear structure in a higher dimensional space back into the dimensions of the original data

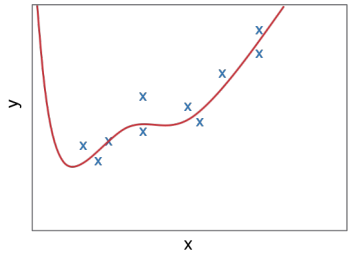
7

Higher-Order Fitting

Order-3 Solution

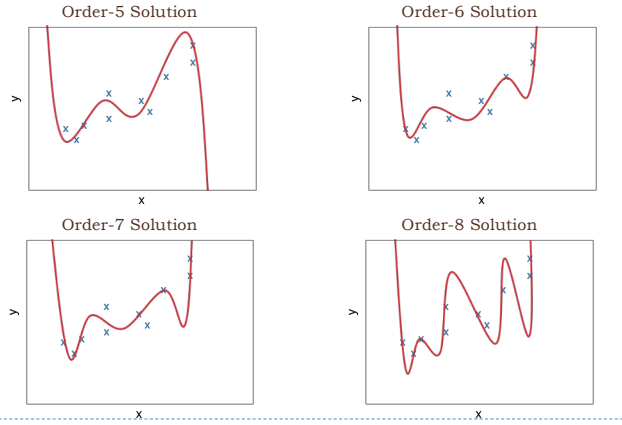


Order-4 Solution



8

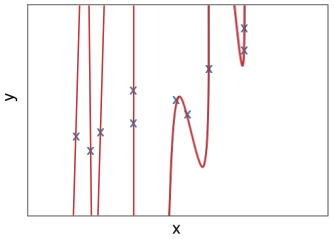
Even Higher-Order Fitting



9

The Risk of Overfitting

- An order-9 solution hits all the data points exactly, but is very "wild" at points that are not given in the data, with high variance
- This is a general problem for learning: if we ***over-train***, we can end up with a function that is very precise on the training data, but is not accurate on new examples



10

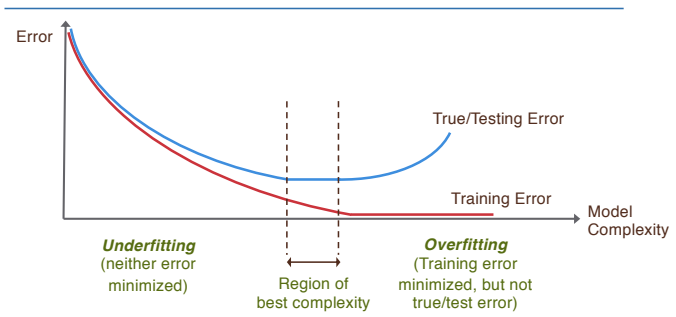
Defining Overfitting

- To precisely understand overfitting, we distinguish between two types of error:
  1. True error: the actual error between the hypothesis and the true function that we want to learn
  2. Training error: the error observed on our training set of examples, during the learning process
- Overfitting is when:
  1. We have a choice between hypotheses, h_1 & h_2
  2. We choose h_1 because it has lowest training error
  3. Choosing h_2 would actually be better, since it will have lowest true error, even if training error is worse
- In general we do not know true error (would essentially need to ***already know*** function we are trying to learn)
  - How then can we ***estimate*** the true error?

11

Model Complexity and Error

- Overfitting often occurs as our models get more complex
  - Higher- and higher-order polynomials for regression
  - Tweaking of parameters to finer and finer degrees of precision



12

Cross-Validation

- We can ***estimate*** our true error by checking how well our function does (on average) when we ***leave out*** some training data, and use it only to test instead
- Leave-one-out cross-validation:

1. For each degree d and k items, we train k different models (a total of $k * d$ tests).
2. For each of the k tests, we take out one example from the input set, and train on all the rest.
3. For each trained model, we test on the one example we left out, and measure the error.
4. We choose the degree d that gives us the lowest mean error on the k tests.

13

An Example of Error Estimation

- For data-set of 10 (*input, output*) pairs, we estimate error using 10 tests:

$$Data = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_{10}, y_{10})\}.$$

Iter	Train-set	Test-set	Train-error	Test-error
1	$Data - \{(\mathbf{x}_1, y_1)\}$	$\{(\mathbf{x}_1, y_1)\}$	0.4928	0.0044
2	$Data - \{(\mathbf{x}_2, y_2)\}$	$\{(\mathbf{x}_2, y_2)\}$	0.1995	0.1869
3	$Data - \{(\mathbf{x}_3, y_3)\}$	$\{(\mathbf{x}_3, y_3)\}$	0.3461	0.0053
4	$Data - \{(\mathbf{x}_4, y_4)\}$	$\{(\mathbf{x}_4, y_4)\}$	0.3887	0.8681
5	$Data - \{(\mathbf{x}_5, y_5)\}$	$\{(\mathbf{x}_5, y_5)\}$	0.2128	0.3439
6	$Data - \{(\mathbf{x}_6, y_6)\}$	$\{(\mathbf{x}_6, y_6)\}$	0.1996	0.1567
7	$Data - \{(\mathbf{x}_7, y_7)\}$	$\{(\mathbf{x}_7, y_7)\}$	0.5707	0.7205
8	$Data - \{(\mathbf{x}_8, y_8)\}$	$\{(\mathbf{x}_8, y_8)\}$	0.2661	0.0203
9	$Data - \{(\mathbf{x}_9, y_9)\}$	$\{(\mathbf{x}_9, y_9)\}$	0.3604	0.2033
10	$Data - \{(\mathbf{x}_{10}, y_{10})\}$	$\{(\mathbf{x}_{10}, y_{10})\}$	0.2138	1.0490
		mean:	0.2188	0.3558

14

An Example of Error Estimation

- By comparing all possible degrees of our function (1–8), we can see that we get the optimal estimated function at degree 2, with overfitting seen at all degrees higher than that:

Degree	Mean Train-error	Mean Test-error
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

Optimal degree: minimizes the estimated error over new examples

Over-fitting: we have minimized the error over training data, but have larger estimated error over new examples

15

More General Cross-Validation

- Leave-one-out validation can be quite costly with large input sets, and so we often test machine learning algorithms in more approximate ways
- k-fold cross-validation is a more granular approach:

1. Divide the input into k different test sets.
2. On each run, remove one of the test sets.
3. Train on the remainder and test on the test set.
4. Average the k results to estimate true error.

16