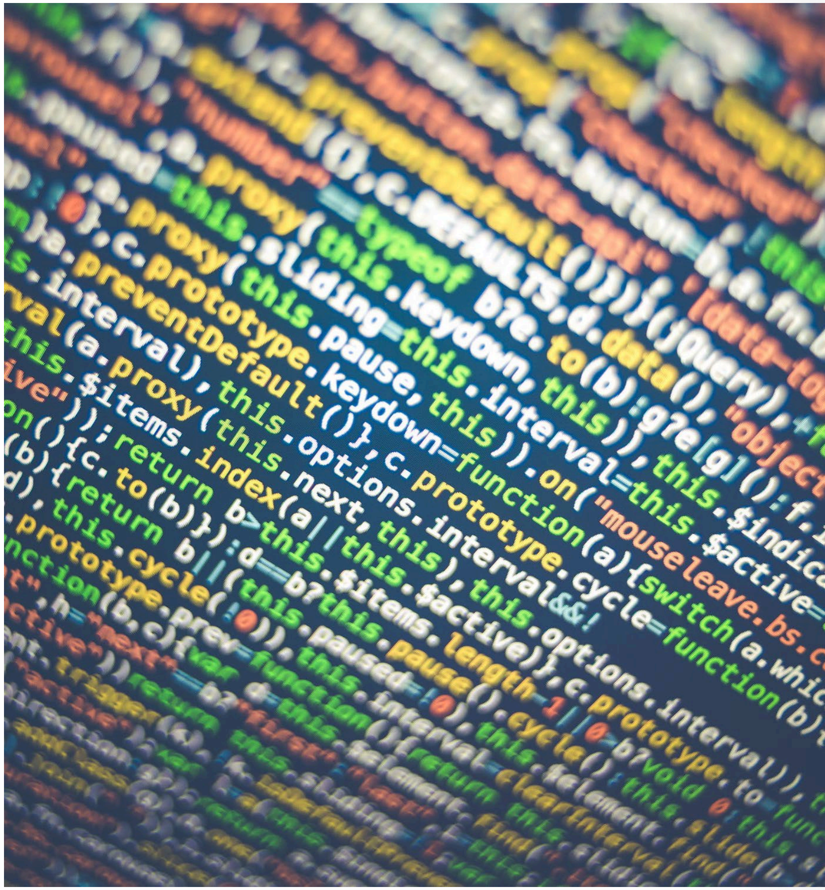
LINEAR REGRESSION

Chih-Chung Hsu (許志仲)
Assistant Professor
ACVLab, Institute of Data Science
National Cheng Kung University
https://cchsu.info



Supervised learning

- Given a set of **<u>training examples</u>**: $x_i = < x_{i1}, x_{i2}, x_{i3}, \ldots, x_{in}, y_i >$

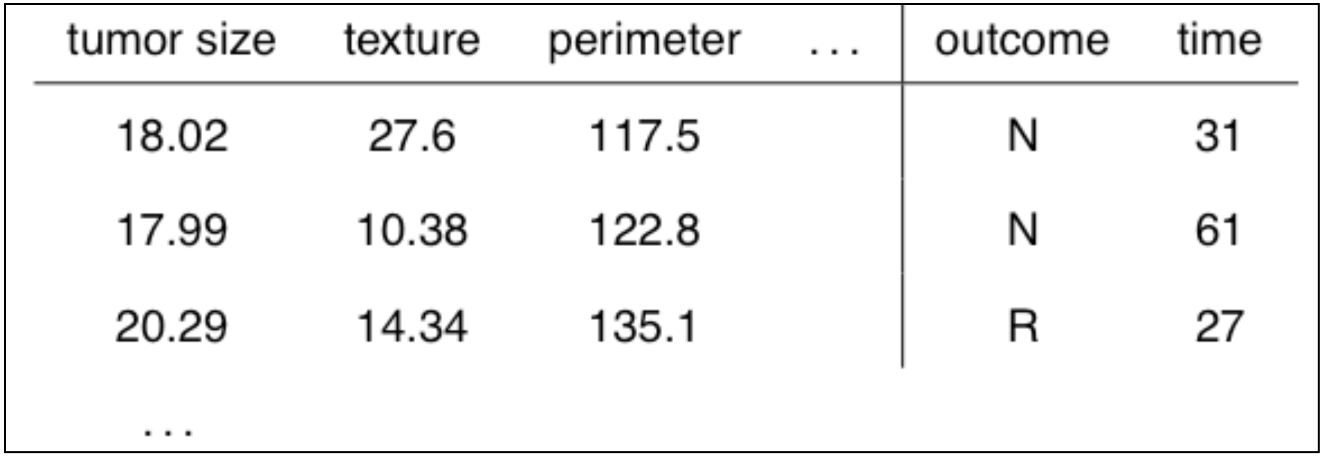
  x_{ij} is the j^{th} feature of the i^{th} example

  y_i is the desired **<u>output</u> (**or **<u>target)</u>** for the i^{th} example.

  X_j denotes the j^{th} feature.

- We want to learn a function $f : X_1 \times X_2 \times \ldots \times X_n \rightarrow Y$

  which maps the input variables onto the output domain.



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
. . .					

Supervised learning

- Given a dataset $X \times Y$, find a function: $f : X \to Y$ such that $f(\mathbf{x})$ is a good predictor for the value of y.
- Formally, f is called the **hypothesis**.
- Output Y can have many types:
  - If $Y = \Re$, this problem is called **regression**.
  - If Y is a finite discrete set, the problem is called **classification**.
  - If Y has 2 elements, the problem is called **binary classification**.

Prediction problems

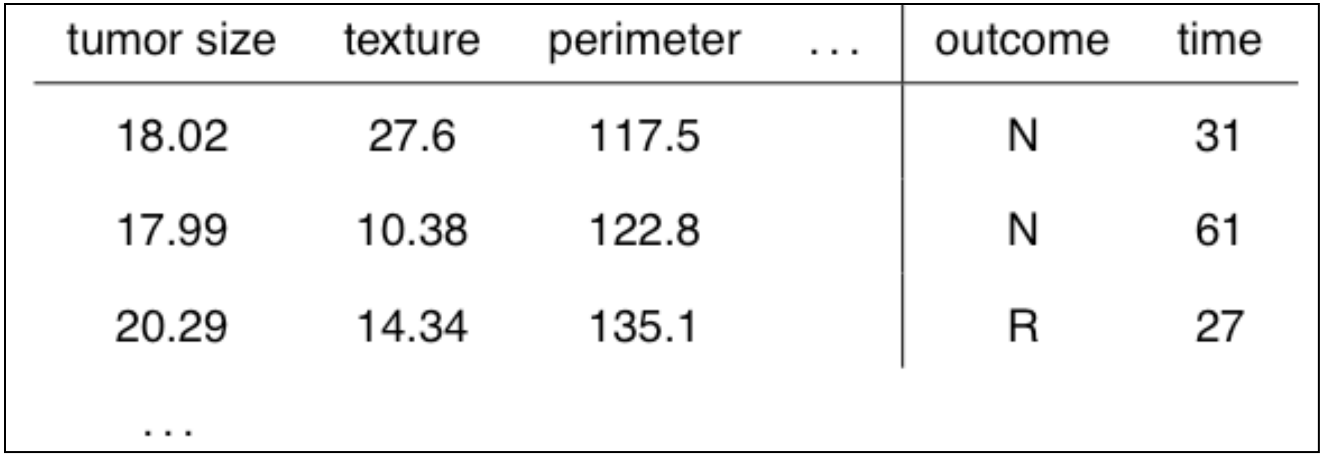
- The problem of predicting tumour recurrence is called:

  classification

- The problem of predicting the time of recurrence is called:

  regression

- Treat them as two separate supervised learning problems.



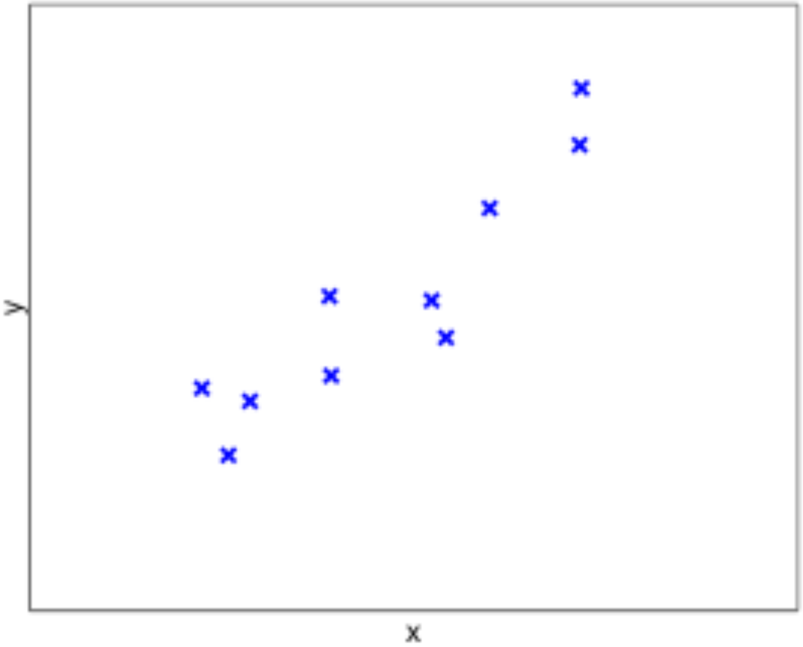
tumor size	texture	perimeter	. . .	outcome	time
18.02	27.6	117.5		N	31
17.99	10.38	122.8		N	61
20.29	14.34	135.1		R	27
. . .					

Variable types

- **Quantitative**, often real number measurements.
  - Assumes that similar measurements are similar in nature.
- **Qualitative**, from a set (categorical, discrete).
  - E.g. {Spam, Not-spam}
- **Ordinal**, also from a discrete set, without metric relation, but that allows ranking.
  - E.g. {first, second, third}

A regression problem

- What <u>hypothesis class</u> should we pick?



Observe x	*Predict* 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
-1.1	-0.12
0.40	1.81
-0.96	-0.83
0.17	0.43

Linear hypothesis

- Suppose *Y* is a **linear function** of ***X***:

$$f_{\mathbf{W}}(\mathbf{X}) = w_0 + w_1\, x_1 + \ldots + w_m\, x_m$$

$$= w_0 + \textstyle\sum_{j=1:m} w_j\, x_j$$

- The w_j are called **parameters** or **weights**.

- To simplify notation, we add an attribute $x_0=1$ to the m other attributes (also called **bias term** or **intercept**).

How should we pick the *weights*?

Least-squares solution method

- The linear regression problem: $f_{\mathbf{w}}(X) \quad = \quad w_0 + \sum_{j=1:m} w_j\, x_j$

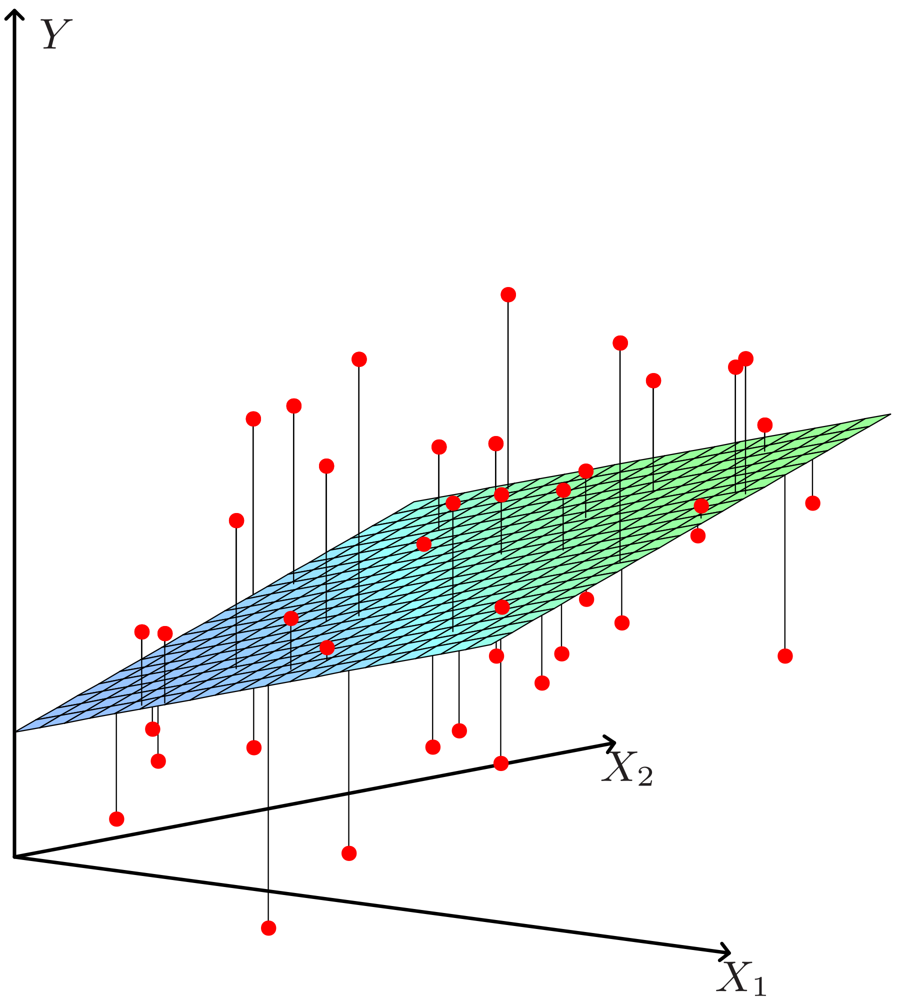
  where m = the dimension of observation space, i.e. number of features.

- **Goal**: Find the **best** linear model given the data.
- Many different possible **evaluation** criteria!
- Most common choice is to find the $\mathbf{w}$ that minimizes:

$$Err(w) = \sum_{i=1:n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

(A note on notation: Here $\mathbf{w}$ and $\mathbf{x}$ are column vectors of size $m+1$.)

Least-squares solution for $X \in \Re^2$



Least-squares solution method

- Re-write in matrix notation: $f_{\mathbf{w}}(X) = X\mathbf{w}$

  $$Err(\mathbf{w}) = (Y - X\mathbf{w})^T(Y - X\mathbf{w})$$

  where X is the $n \times m$ matrix of input data,
  Y is the $n \times 1$ vector of output data,
  $\mathbf{w}$ is the $m \times 1$ vector of weights.

- To minimize, take the derivative w.r.t. $\mathbf{w}$:

  $$\partial Err(\mathbf{w})/\partial \mathbf{w} = -2\, X^T (Y - X\mathbf{w})$$

  – You get a system of m equations with m unknowns.

- Set these equations to 0: $X^T (Y - X\mathbf{w}) = 0$

  – *Remember that derivative has to be 0 at a minimum of Err(w)*

Least-squares solution method

- We want to solve for ***w***: $X^T (Y - X\mathbf{w}) = 0$

- Try a little algebra: $X^T Y = X^T X \mathbf{w}$

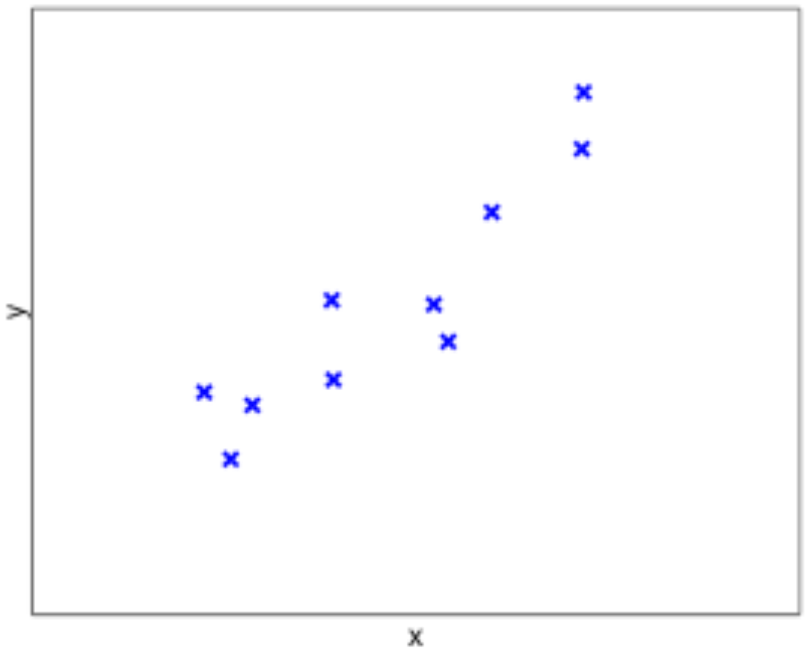
  $\hat{\mathbf{w}} = (X^T X)^{-1} X^T Y$

  ($\hat{\mathbf{w}}$ denotes the estimated weights)

- Train set predictions: $\hat{Y} = X\hat{\mathbf{w}} = X (X^T X)^{-1} X^T Y$

- Predict new data $X' \rightarrow Y'$: $Y' = X'\hat{\mathbf{w}} = X' (X^T X)^{-1} X^T Y$

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

What is a plausible estimate of ***w*** ?

Try it!

Data matrices

$$X^T X =$$

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

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

Data matrices

$$X^T Y =$$

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

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

Solving the problem

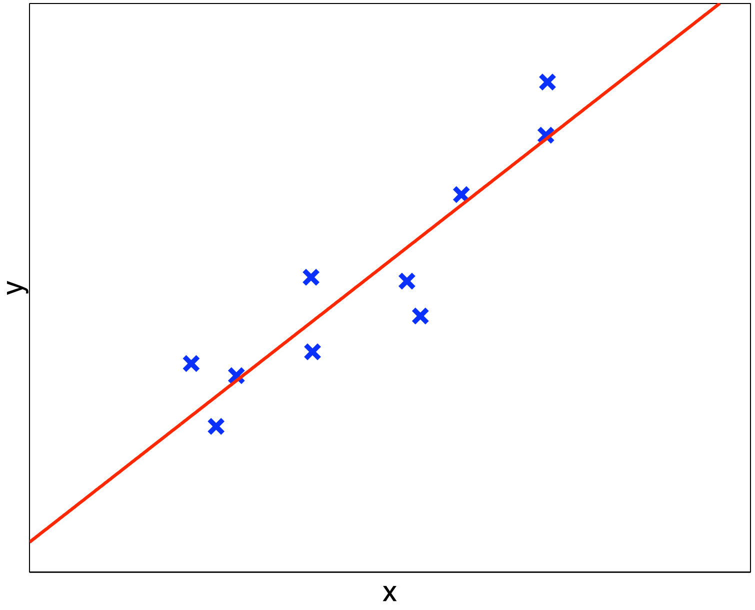
$$\mathbf{w} = (X^T X)^{-1} X^T Y = \begin{bmatrix} 4.95 & -1.39 \\ -1.39 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 6.49 \\ 8.34 \end{bmatrix} = \begin{bmatrix} 1.60 \\ 1.05 \end{bmatrix}$$

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

Solving the problem

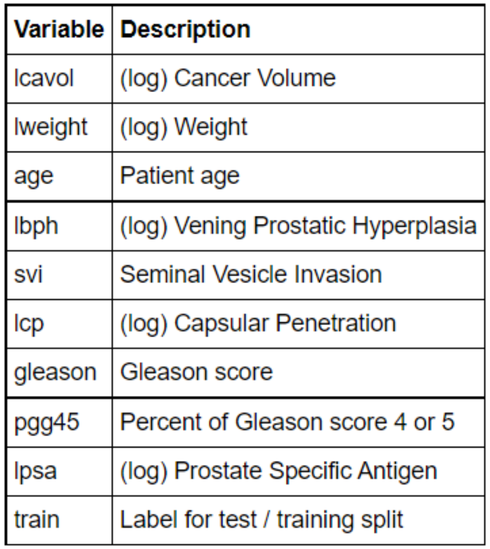
$$\mathbf{w} = (X^T X)^{-1} X^T Y = \begin{bmatrix} 4.95 & -1.39 \\ -1.39 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 6.49 \\ 8.34 \end{bmatrix} = \begin{bmatrix} 1.60 \\ 1.05 \end{bmatrix}$$

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

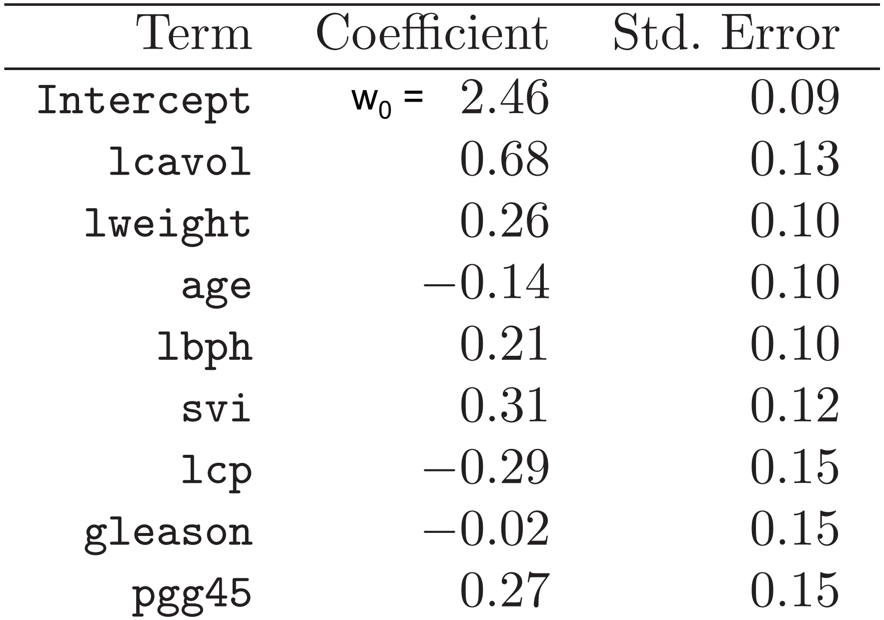


Interpreting the solution

- Linear fit for a prostate cancer dataset
  - Features *X* = {lcavol , lweight, age, lbph, svi, lcp, gleason, pgg45}
  - Output *y* = level of PSA (an enzyme which is elevated with cancer).
  - High coefficient weight (in absolute value) = important for prediction.



Variable	Description
lcavol	(log) Cancer Volume
lweight	(log) Weight
age	Patient age
lbph	(log) Vening Prostatic Hyperplasia
svi	Seminal Vesicle Invasion
lcp	(log) Capsular Penetration
gleason	Gleason score
pgg45	Percent of Gleason score 4 or 5
lpsa	(log) Prostate Specific Antigen
train	Label for test / training split



Term	Coefficient	Std. Error
Intercept	$w_0 =$ 2.46	0.09
lcavol	0.68	0.13
lweight	0.26	0.10
age	−0.14	0.10
lbph	0.21	0.10
svi	0.31	0.12
lcp	−0.29	0.15
gleason	−0.02	0.15
pgg45	0.27	0.15

Interpreting the solution

- Caveat: data should be in same range
- If we change unit for age from years to months, we expect the optimal weight to be 12x as low (so predictions don't change)
- Doesn't mean age became 12x less relevant!
- Can **normalize** data to make range similar
  - E.g. subtract average and divide by standard deviation
- More principled approach in next lecture

Computational cost of linear regression

- What operations are necessary?

Computational cost of linear regression

- What operations are necessary?
  - Overall: 1 matrix inversion + 3 matrix multiplications
  - X^TX (other matrix multiplications require fewer operations.)
    - X^T is mxn and X is nxm, so we need nm^2 operations.
  - $(X^TX)^{-1}$
    - X^TX is mxm, so we need m^3 operations.

Computational cost of linear regression

- What operations are necessary?
  - Overall: 1 matrix inversion + 3 matrix multiplications
  - X^TX (other matrix multiplications require fewer operations.)
    - X^T is mxn and X is nxm, so we need nm^2 operations.
  - $(X^TX)^{-1}$
    - X^TX is mxm, so we need m^3 operations.

- We can do linear regression in polynomial time, but handling large datasets (many examples, many features) can be problematic.

An alternative for minimizing mean-squared error (MSE)

- Recall the least-square solution: $\hat{\boldsymbol{w}} = (X^T X)^{-1} X^T Y$
- What if X is too big to compute this explicitly (e.g. $m \sim 10^6$)?

An alternative for minimizing mean-squared error (MSE)

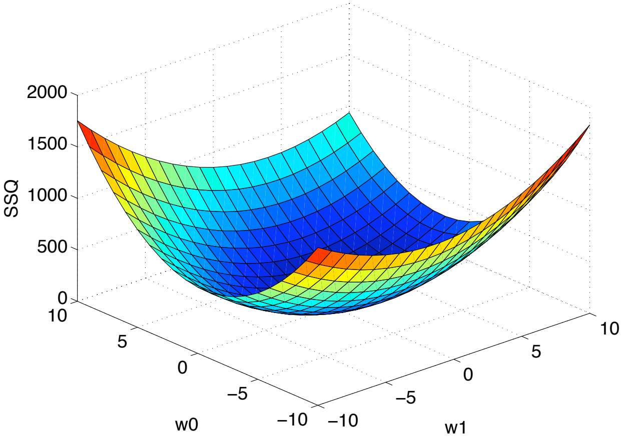
- Recall the least-square solution: $\hat{\mathbf{w}} = (X^T X)^{-1} X^T Y$
- What if X is too big to compute this explicitly (e.g. $m \sim 10^6$)?
- Go back to the gradient step: $Err(\mathbf{w}) = (Y - X\mathbf{w})^T (Y - X\mathbf{w})$

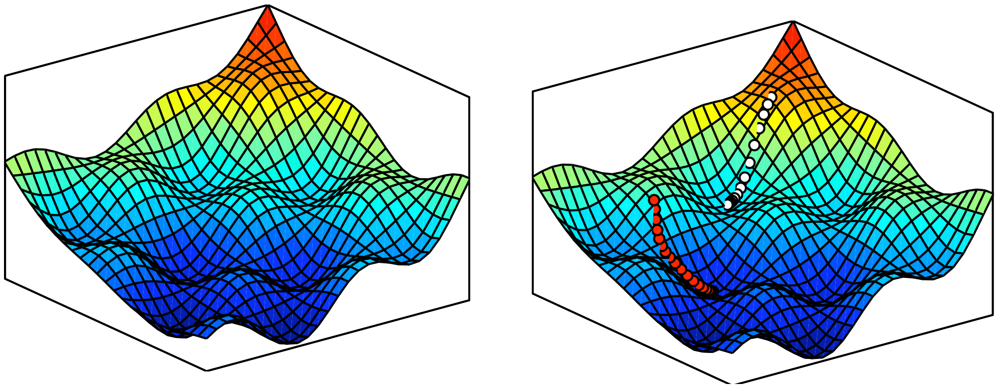
$$\partial Err(\mathbf{w})/\partial \mathbf{w} = -2\, X^T (Y - X\mathbf{w})$$

$$\partial Err(\mathbf{w})/\partial \mathbf{w} = 2(X^T X\mathbf{w} - X^T Y)$$

Gradient-descent solution for MSE

- Consider the error function:





- The gradient of the error is a vector indicating the direction to the minimum point.
- Instead of directly finding that minimum (using the closed-form equation), we can take small steps towards the minimum.

Gradient-descent solution for MSE

- We want to produce a sequence of weight solutions, $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2 \ldots,$ such that: $Err(\mathbf{w}_0) > Err(\mathbf{w}_1) > Err(\mathbf{w}_2) > \ldots$

Gradient-descent solution for MSE

- We want to produce a sequence of weight solutions, $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2\ldots,$ such that: $Err(\mathbf{w}_0) > Err(\mathbf{w}_1) > Err(\mathbf{w}_2) > \ldots$

- The algorithm:

```
Given an initial weight vector $\mathbf{w}_0$,
Do for $k=1, 2, \ldots$
        $\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \, \partial Err(\mathbf{w}_k)/\partial \mathbf{w}_k$
End when $|\mathbf{w}_{k+1}-\mathbf{w}_k| < \varepsilon$
```

- Parameter $\alpha_k>0$ is the step-size (or learning rate) for iteration k.

Convergence

- Convergence depends in part on the α_k.

- If steps are too **large**: the $\boldsymbol{w}_k$ may oscillate forever.
  - This suggests that $\alpha_k \to 0$ as $k \to \infty$.

- If steps are too **small**: the $\boldsymbol{w}_k$ may not move far enough to reach a local minimum.

Robbins-Monroe conditions

- The α_k are a Robbins-Monroe sequence if:

  $$\sum_{k=0:\infty} \alpha_k = \infty$$

  $$\sum_{k=0:\infty} \alpha_k^2 < \infty$$

- These conditions are sufficient to ensure convergence of the $\boldsymbol{w}_k$ to a **local minimum** of the error function.

Robbins-Monroe conditions

- The α_k are a Robbins-Monroe sequence if:

  $$\sum_{k=0:\infty} \alpha_k = \infty$$

  $$\sum_{k=0:\infty} \alpha_k^2 < \infty$$

- These conditions are sufficient to ensure convergence of the $\boldsymbol{w}_k$ to a **local minimum** of the error function.

E.g. $\alpha_k = 1 / (k + 1)$ (averaging)

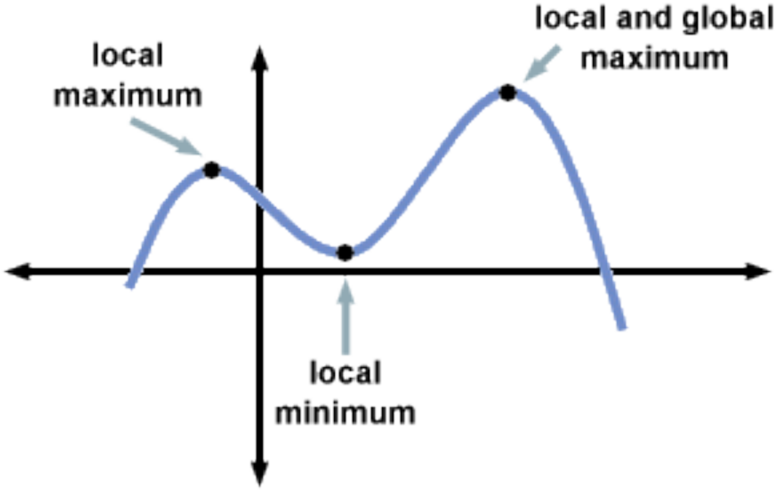
E.g. $\alpha_k = 1/2$ for $k = 1, \ldots, T$

$\alpha_k = 1/2^2$ for $k = T+1, \ldots, (T+1)+2T$

etc.

Local minima

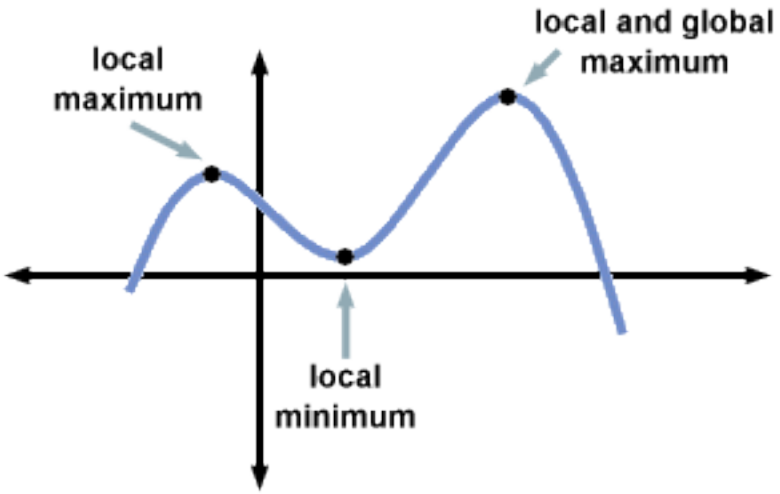
- Convergence is **NOT** to a global minimum, only to local minimum.



- The blue line represents the error function. There is *no guarantee* regarding the amount of error of the weight vector found by gradient descent, compared to the globally optimal solution.

Local minima

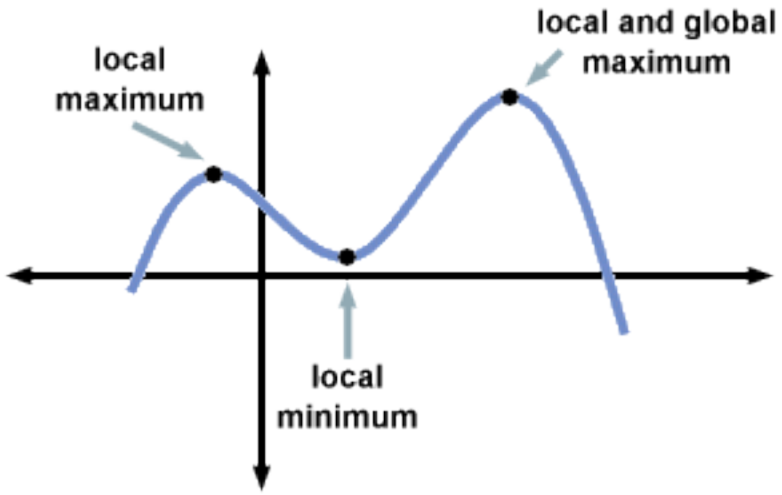
- Convergence is **NOT** to a global minimum, only to local minimum.



- For linear function approximations using Least-Mean Squares (LMS) error, this is not an issue: only **ONE** global minimum!
  - Local minima affects many other function approximators.

Local minima

- Convergence is **<u>NOT</u>** to a global minimum, only to local minimum.



- For linear function approximations using Least-Mean Squares (LMS) error, this is not an issue: only **ONE** global minimum!
  - Local minima affects many other function approximators.
- *<u>Repeated random restarts</u>* can help (in all cases of gradient search).

Basic least-squares solution method

- Recall the least-square solution: $\hat{\boldsymbol{w}} = (X^T X)^{-1} X^T Y$

- Assuming for now that X is reasonably small so computation and memory are not a problem. Can we always evaluate this?

Basic least-squares solution method

- Recall the least-square solution: $\hat{w} = (X^T X)^{-1} X^T Y$
- Assuming for now that X is reasonably small so computation and memory are not a problem. Can we always evaluate this?
- To have a unique solution, we need $X^T X$ to be nonsingular. That means X must have full column rank (i.e. no features can be expressed using other features.)

Exercise: What if X does not have full column rank? When would this happen? Design an example. Try to solve it.

Dealing with difficult cases of $(X^T X)^{-1}$

- **Case #1**: The weights are not uniquely defined.

  Solution: Re-code or drop some redundant columns of X.

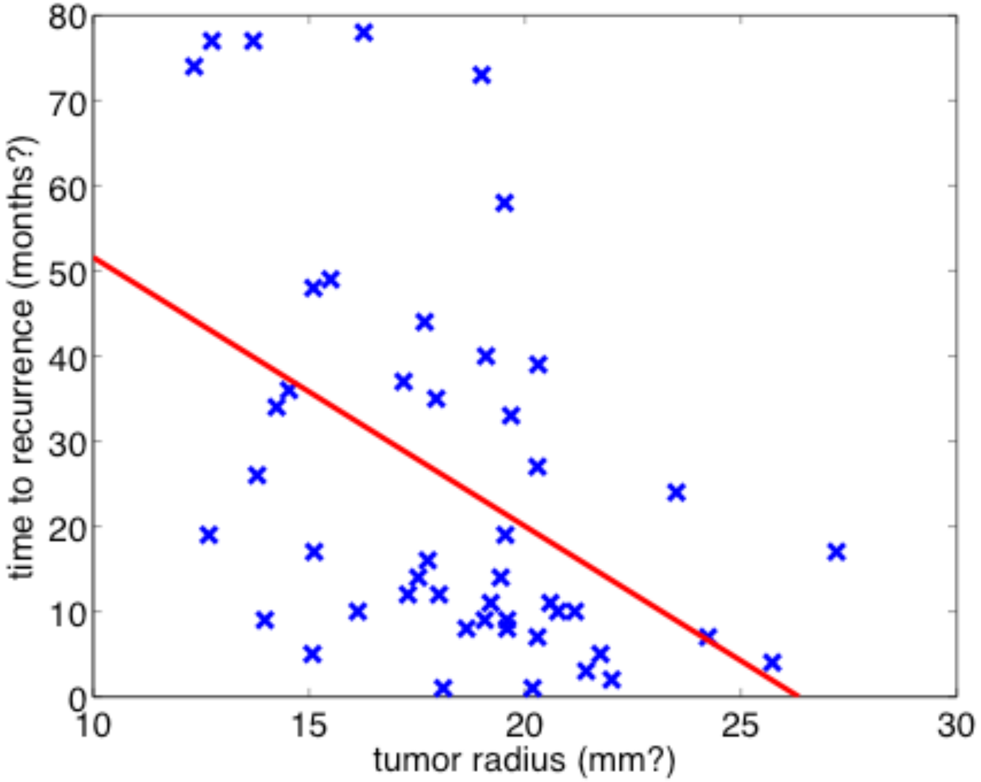
- **Case #2**: The number of features/weights (m) exceeds the number of training examples (n).

  Solution: Reduce the number of features using various techniques (to be studied later.)

Predicting recurrence time from tumor size

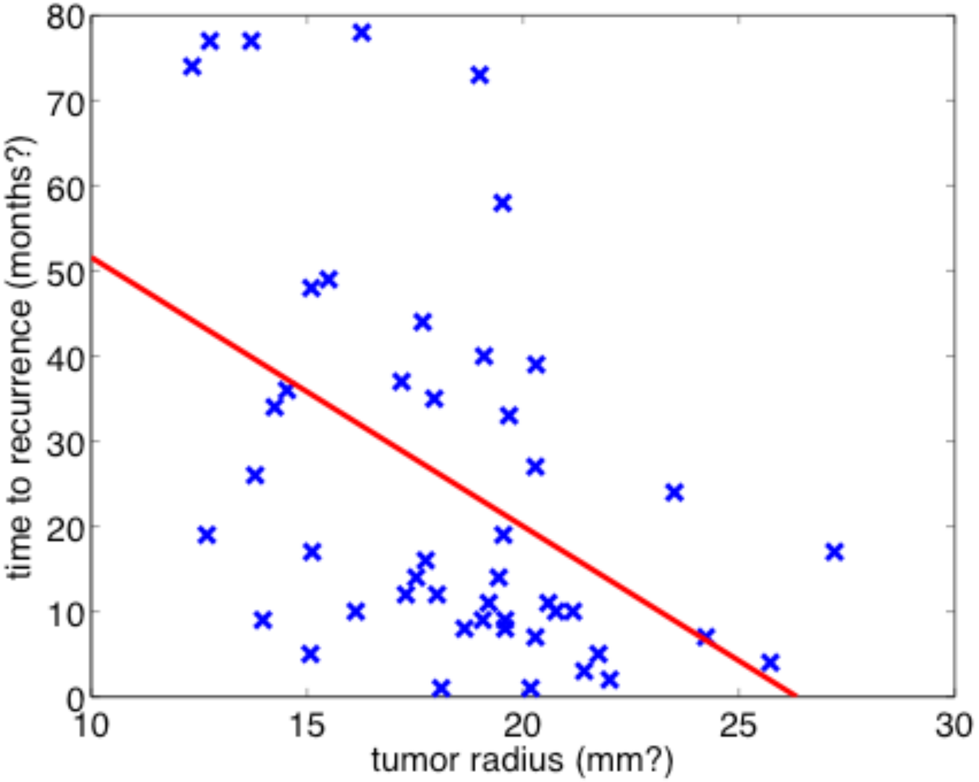
This function looks complicated, and a linear hypothesis does not seem very good.

What should we do?



Predicting recurrence time from tumor size

This function looks complicated, and a linear hypothesis does not seem very good.



What should we do?

- *Pick a better function?*
- *Use more features?*
- *Get more data?*

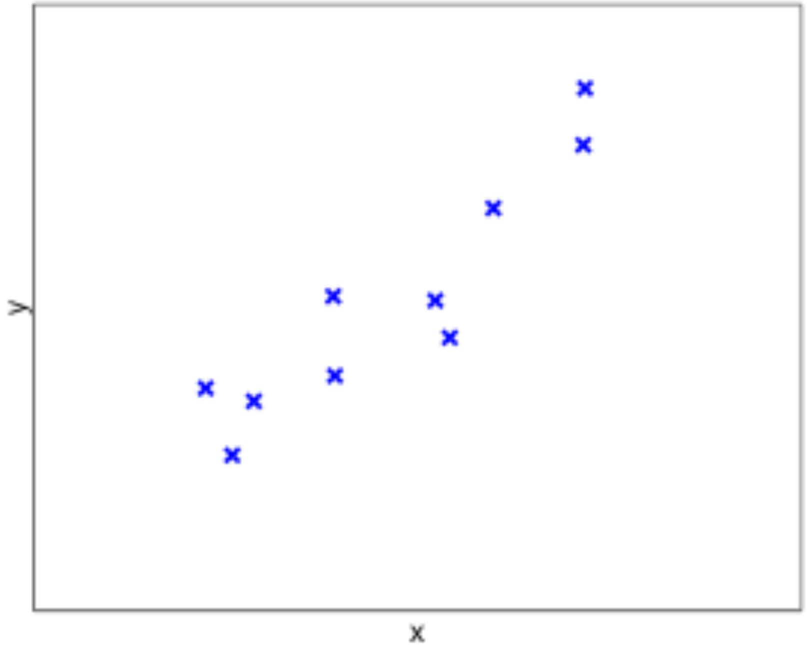
Input variables for linear regression

- Original quantitative variables $X_1, \ldots, X_m$
- Transformations of variables, e.g. $X_{m+1} = log(X_i)$
- Basis expansions, e.g. $X_{m+1} = X_i^2$, $X_{m+2} = X_i^3, \ldots$
- Interaction terms, e.g. $X_{m+1} = X_i X_j$
- Numeric coding of qualitative variables, e.g. $X_{m+1} = 1$ if X_i is true and 0 otherwise.

In all cases, we can add $X_{m+1}, \ldots, X_{m+k}$ to the list of original variables and perform the linear regression.

Example of linear regression with polynomial terms

$$f_w(x) \quad = \quad w_0 + w_1\, x + w_2\, x^2$$

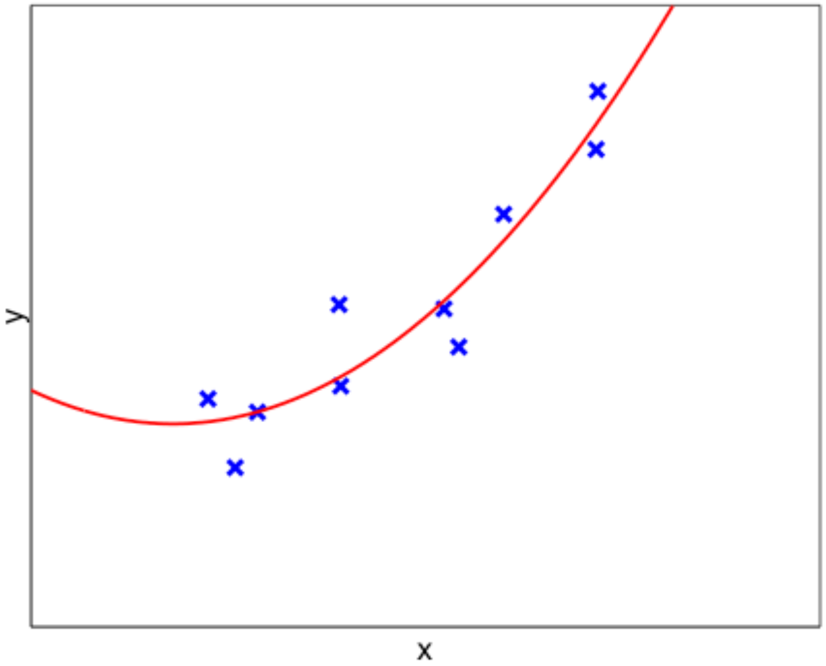


$$X = \begin{array}{c}
\begin{array}{ccc} x^2 & x & \end{array} \\
\begin{bmatrix}
0.75 & 0.86 & 1 \\
0.01 & 0.09 & 1 \\
0.73 & -0.85 & 1 \\
0.76 & 0.87 & 1 \\
0.19 & -0.44 & 1 \\
0.18 & -0.43 & 1 \\
1.22 & -1.10 & 1 \\
0.16 & 0.40 & 1 \\
0.93 & -0.96 & 1 \\
0.03 & 0.17 & 1
\end{bmatrix}
\end{array}
\qquad
Y = \begin{bmatrix}
2.49 \\
0.83 \\
-0.25 \\
3.10 \\
0.87 \\
0.02 \\
-0.12 \\
1.81 \\
-0.83 \\
0.43
\end{bmatrix}$$

Solving the problem

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

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



Compared to *y* = 1.6*x* + 1.05 for the order-1 polynomial.

Input variables for linear regression

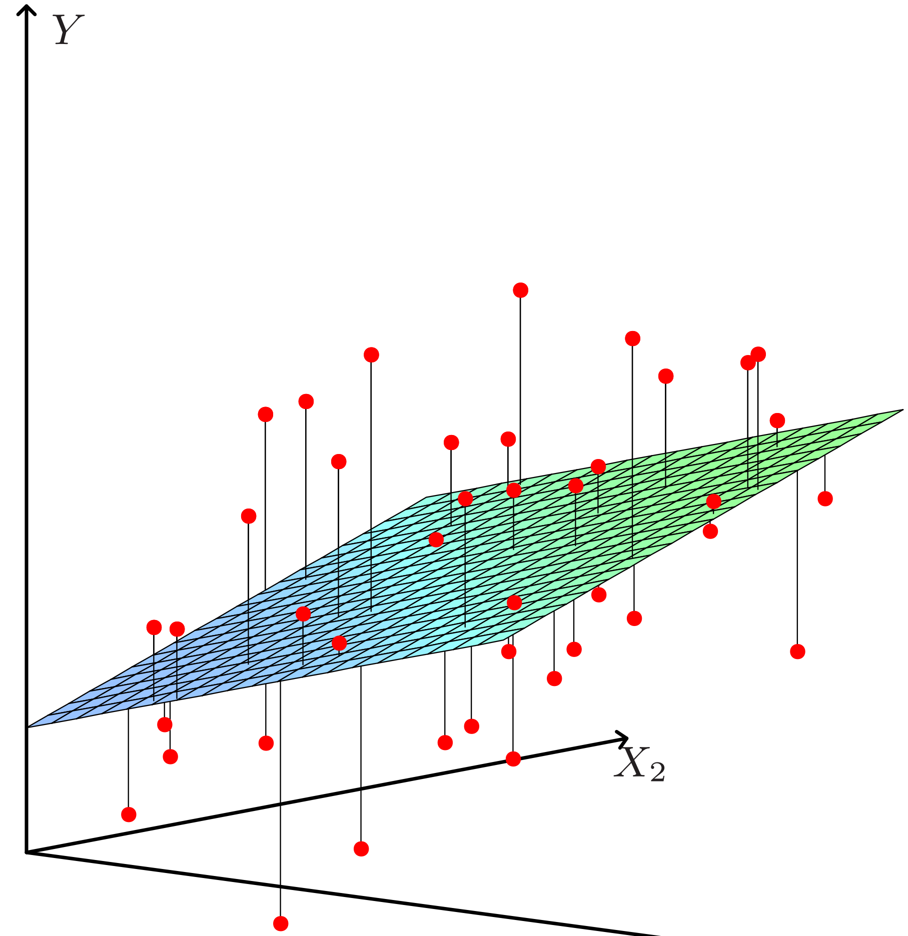
How to choose input variables?

- Propose different strategies, then perform model selection using cross validation (more details later)
- Add many transformation to the set of features, then perform feature selection or dimension reduction (more details later)
- Use problem specific insights:
  - Say, predict displacement of falling option as function of time
  - From physics, know that $s=gt^2$
  - In that case, use squared transformation of t (input variable is t^2)

What you should know

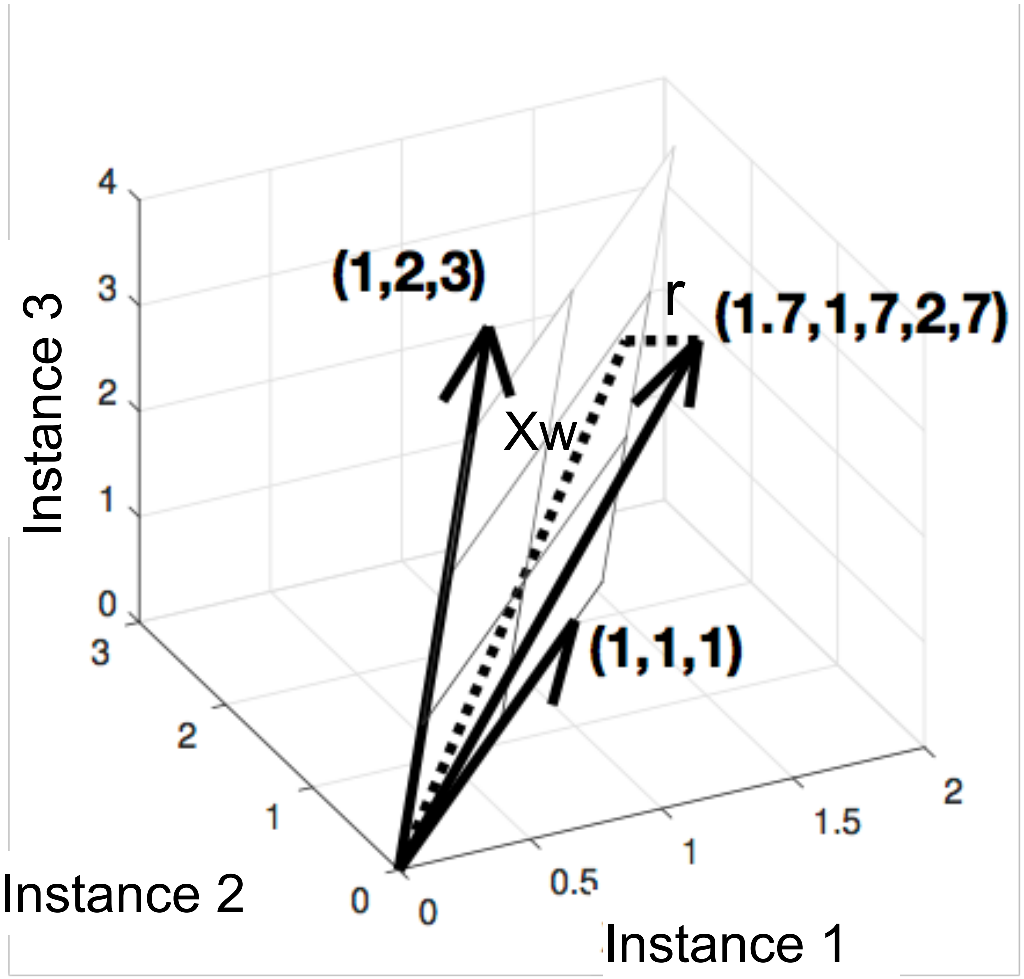
- Definition and characteristics of a supervised learning problem.
- Linear regression (hypothesis class, cost function).
- Closed-form least-squares solution method (algorithm, computational complexity, stability issues).
- Gradient descent method (algorithm, properties).

Weight space view



Instance space view (Geometric view)

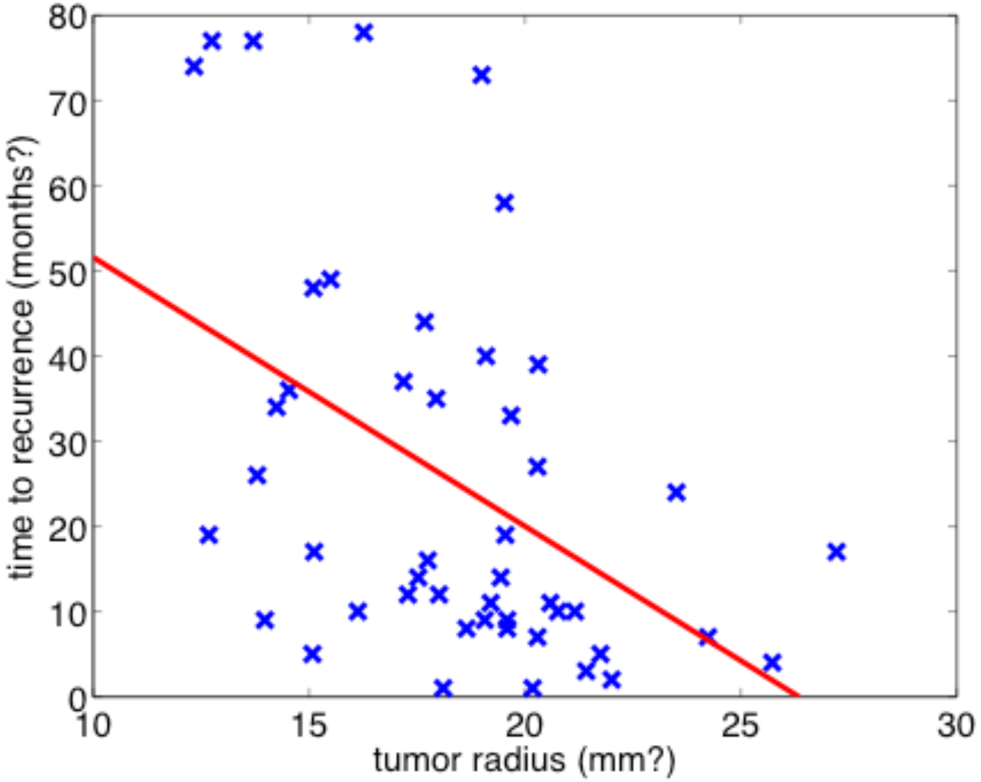
$$\overset{X}{\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \approx \overset{y}{\begin{bmatrix} 1.7 \\ 1.7 \\ 2.7 \end{bmatrix}}$$



Predicting recurrence time from tumor size

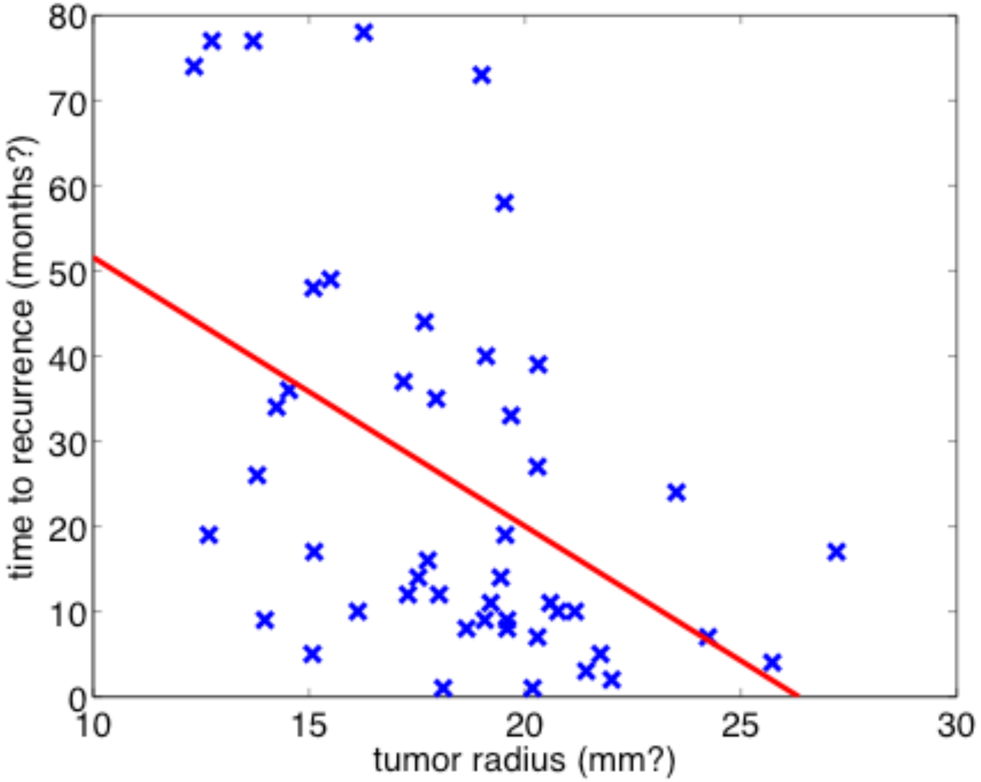
This function looks complicated, and a linear hypothesis does not seem very good.

What should we do?



Predicting recurrence time from tumor size

This function looks complicated, and a linear hypothesis does not seem very good.



What should we do?

- *Pick a better function?*
- *Use more features?*
- *Get more data?*

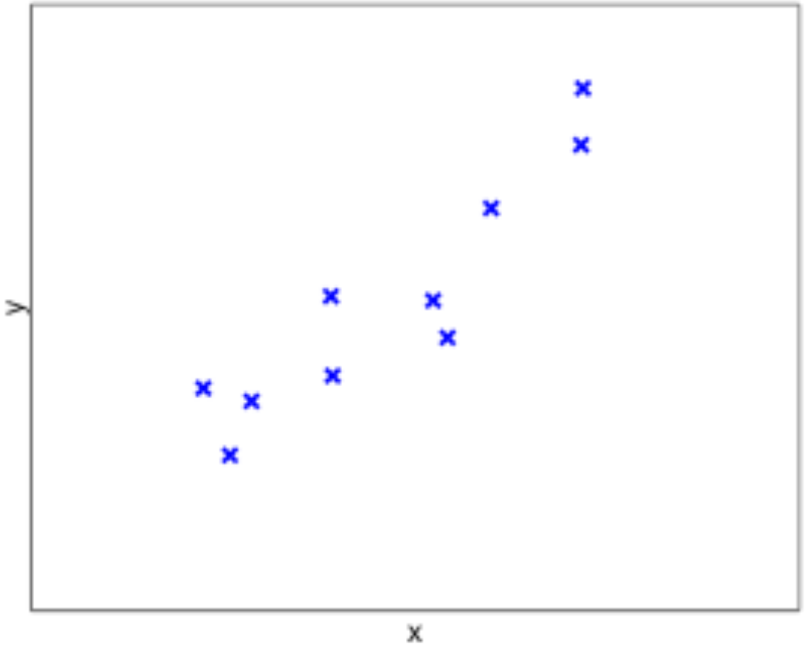
Input variables for linear regression

- Original quantitative variables $X_1, \ldots, X_m$
- Transformations of variables, e.g. $X_{m+1} = log(X_i)$
- Basis expansions, e.g. $X_{m+1} = X_i^2$, $X_{m+2} = X_i^3, \ldots$
- Interaction terms, e.g. $X_{m+1} = X_i X_j$
- Numeric coding of qualitative variables, e.g. $X_{m+1} = 1$ if X_i is true and 0 otherwise.

In all cases, we can add $X_{m+1}, \ldots, X_{m+k}$ to the list of original variables and perform the linear regression.

Example of linear regression with polynomial terms

$$f_w(x) \quad = \quad w_0 + w_1\, x + w_2\, x^2$$

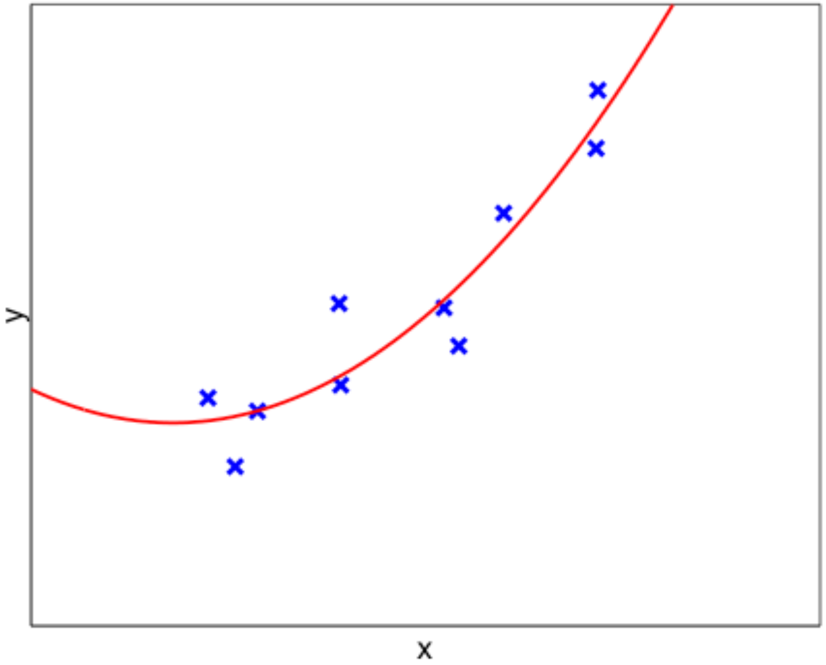


$$\begin{array}{cc} & \begin{array}{ccc} x^2 & x & \end{array} \\ X = & \left[\begin{array}{ccc} 0.75 & 0.86 & 1 \\ 0.01 & 0.09 & 1 \\ 0.73 & -0.85 & 1 \\ 0.76 & 0.87 & 1 \\ 0.19 & -0.44 & 1 \\ 0.18 & -0.43 & 1 \\ 1.22 & -1.10 & 1 \\ 0.16 & 0.40 & 1 \\ 0.93 & -0.96 & 1 \\ 0.03 & 0.17 & 1 \end{array}\right] \end{array} \qquad Y = \left[\begin{array}{c} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{array}\right]$$

Solving the problem

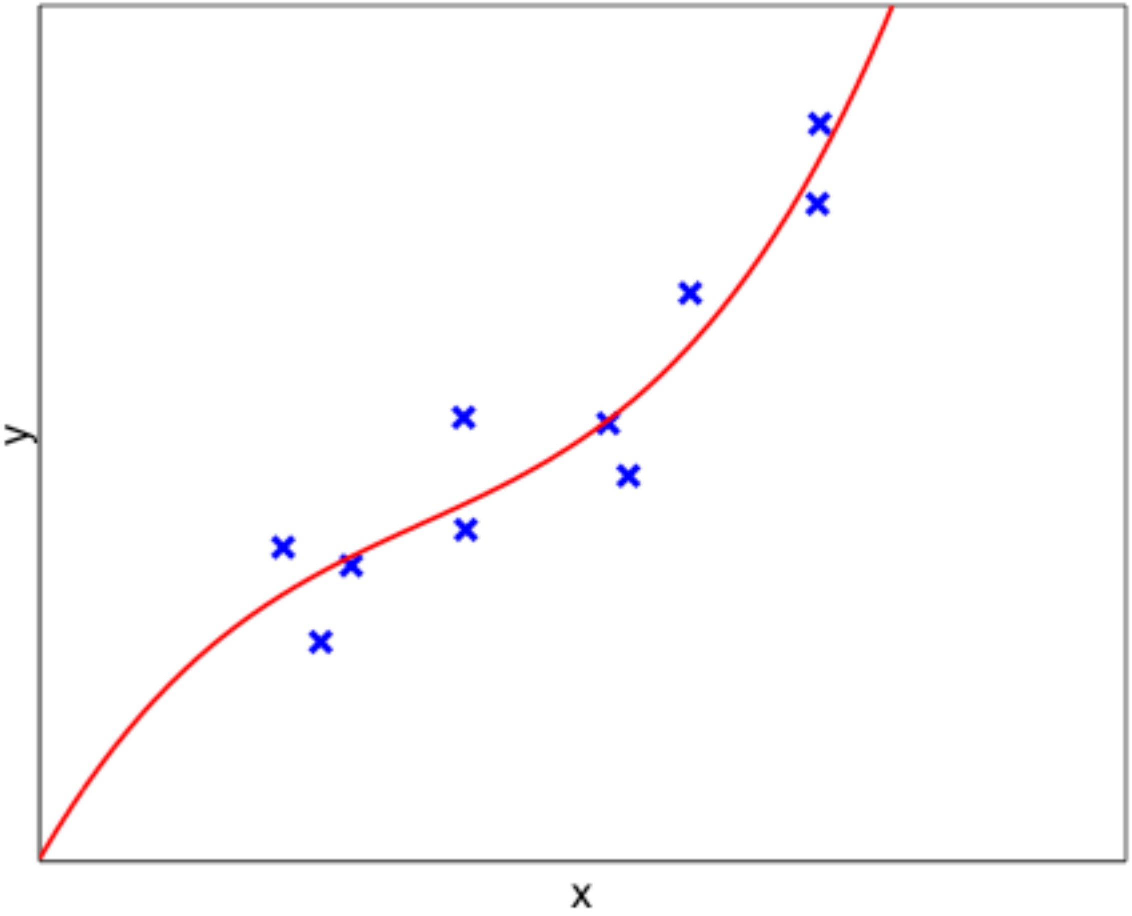
$$\mathbf{w} = (X^T X)^{-1} X^T Y = \left[\begin{array}{ccc} 4.11 & -1.64 & 4.95 \\ -1.64 & 4.95 & -1.39 \\ 4.95 & -1.39 & 10 \end{array}\right]^{-1} \left[\begin{array}{c} 3.60 \\ 6.49 \\ 8.34 \end{array}\right] = \left[\begin{array}{c} 0.68 \\ 1.74 \\ 0.73 \end{array}\right]$$

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

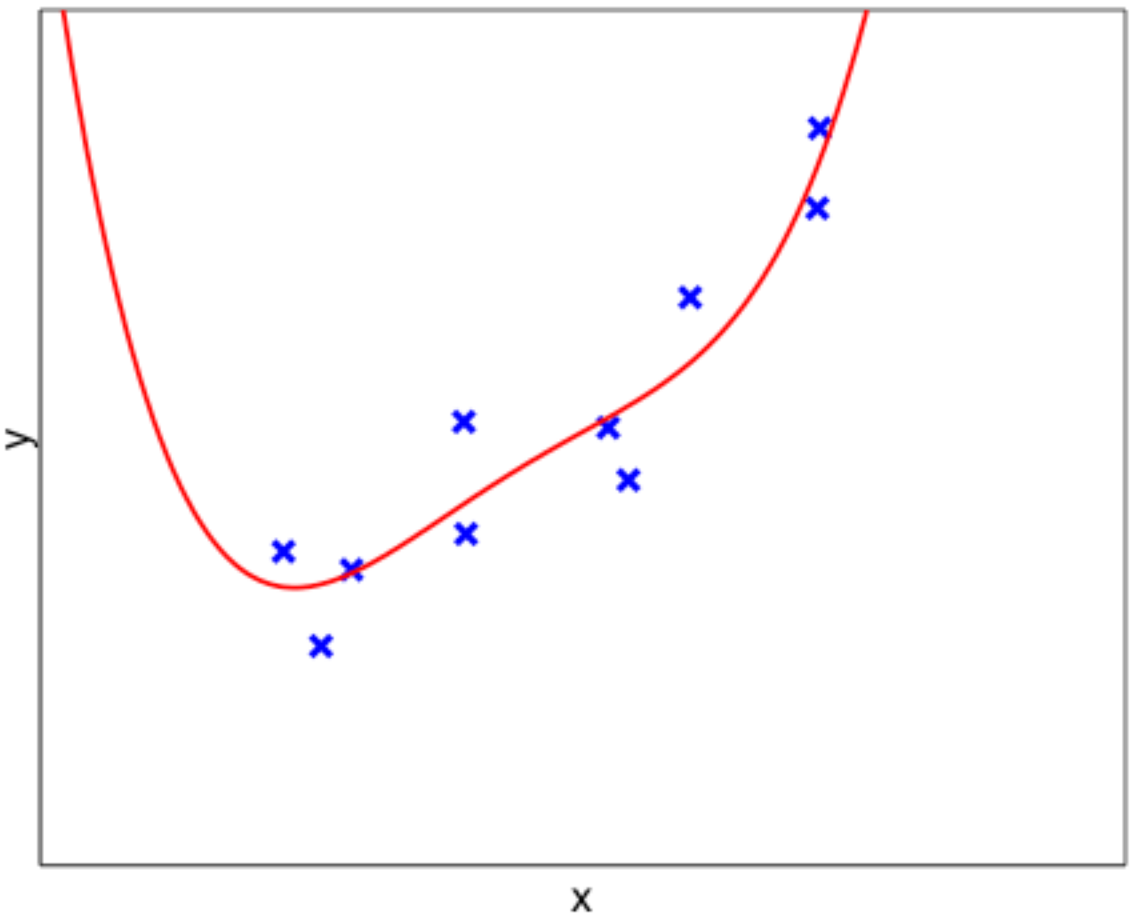


Compared to *y* = 1.6*x* + 1.05 for the order-1 polynomial.

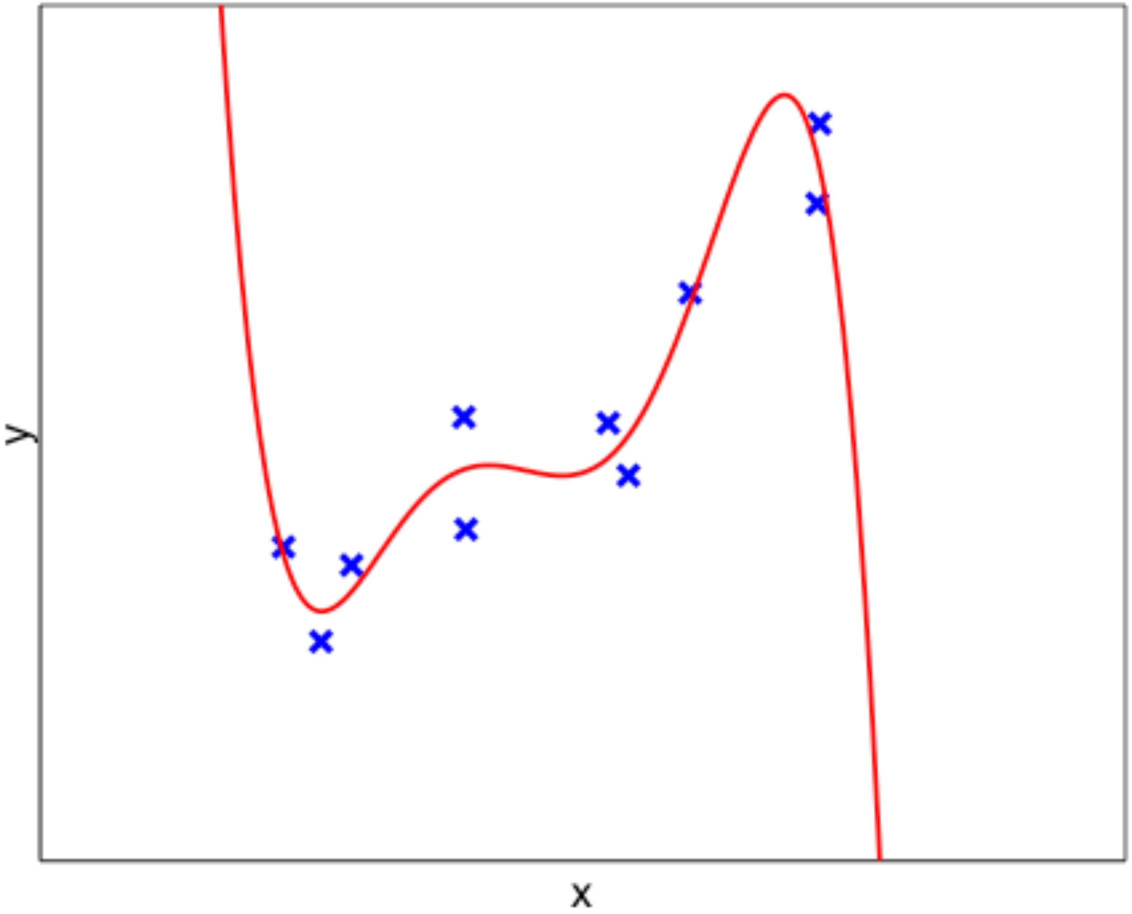
Order-3 fit: Is this better?



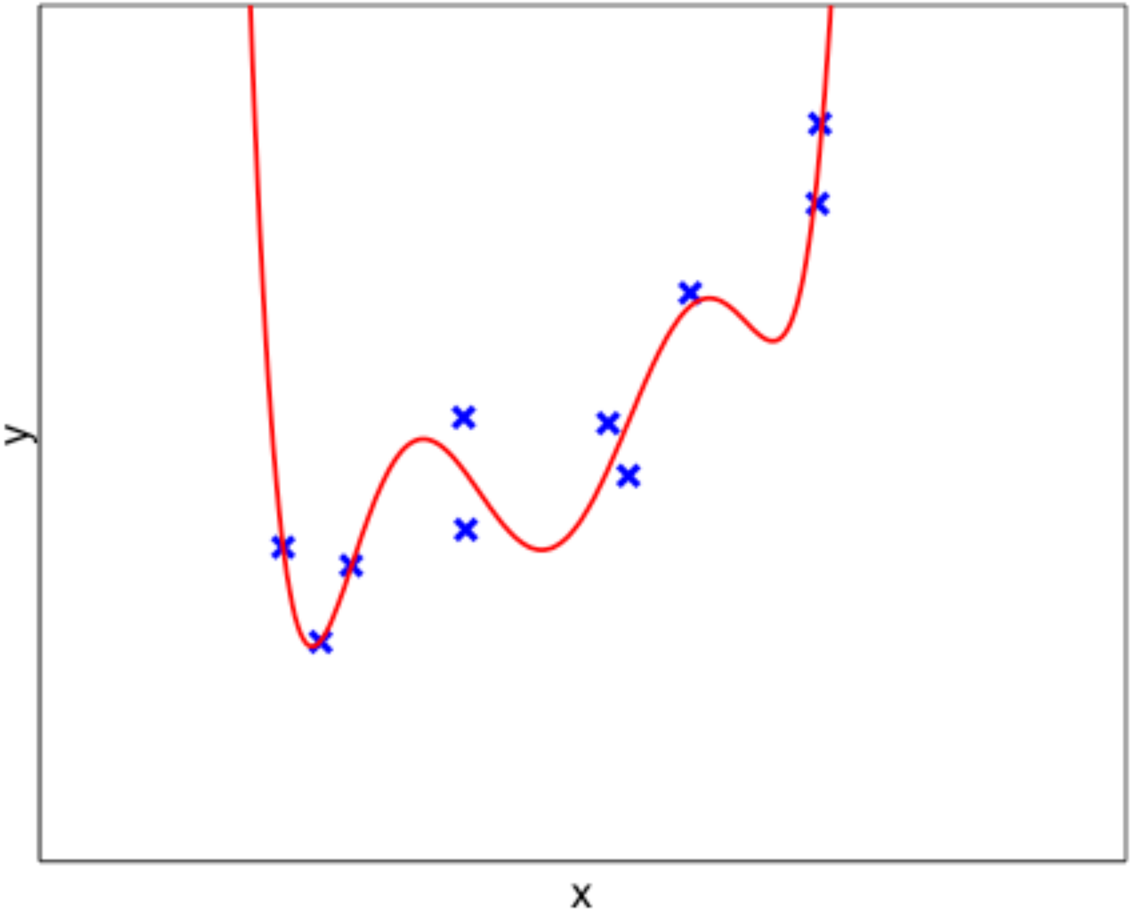
Order-4 fit



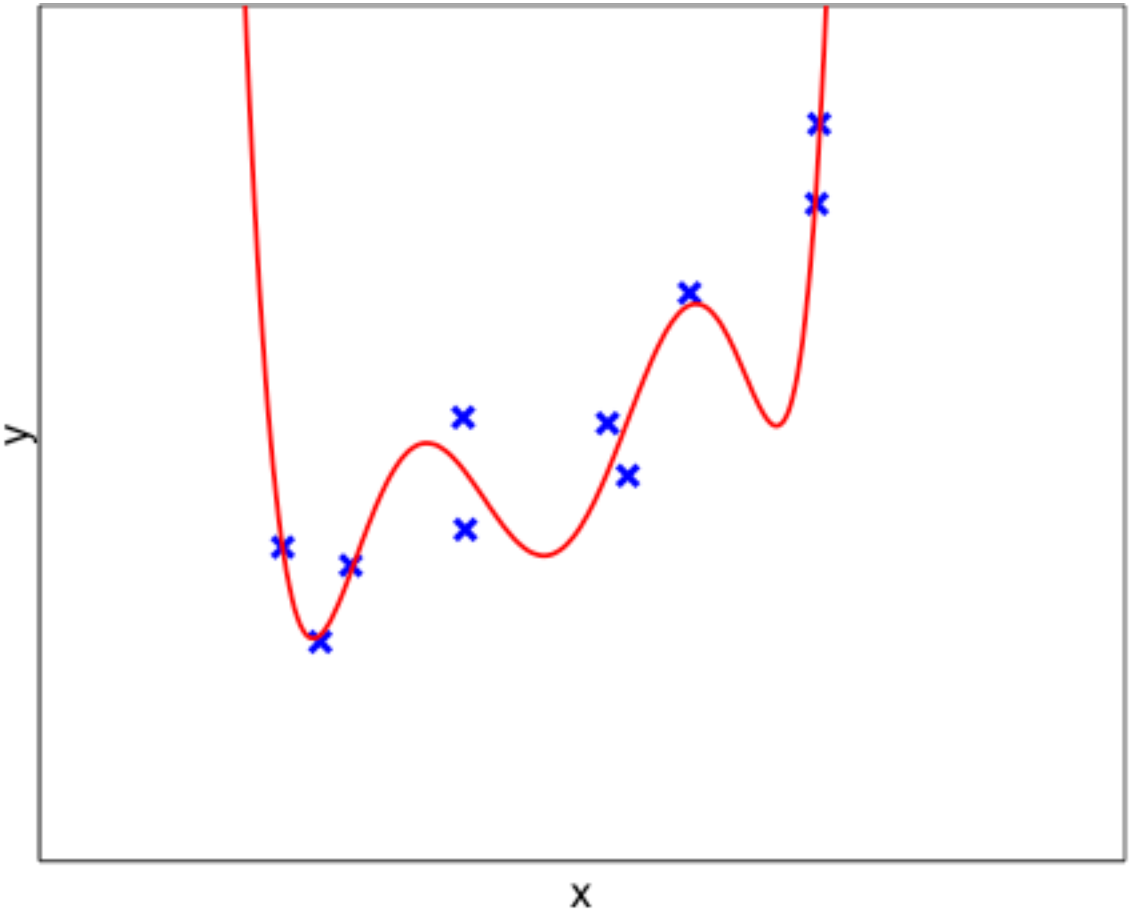
Order-5 fit



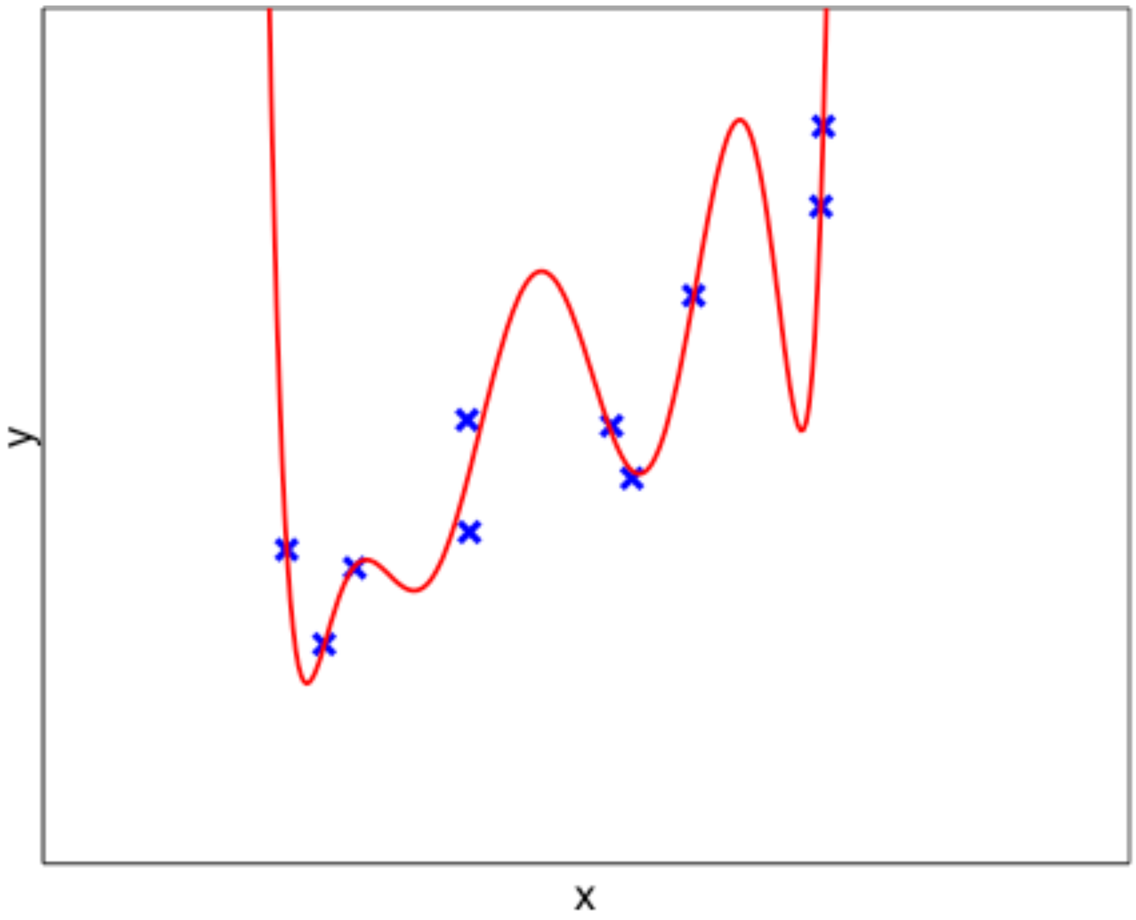
Order-6 fit



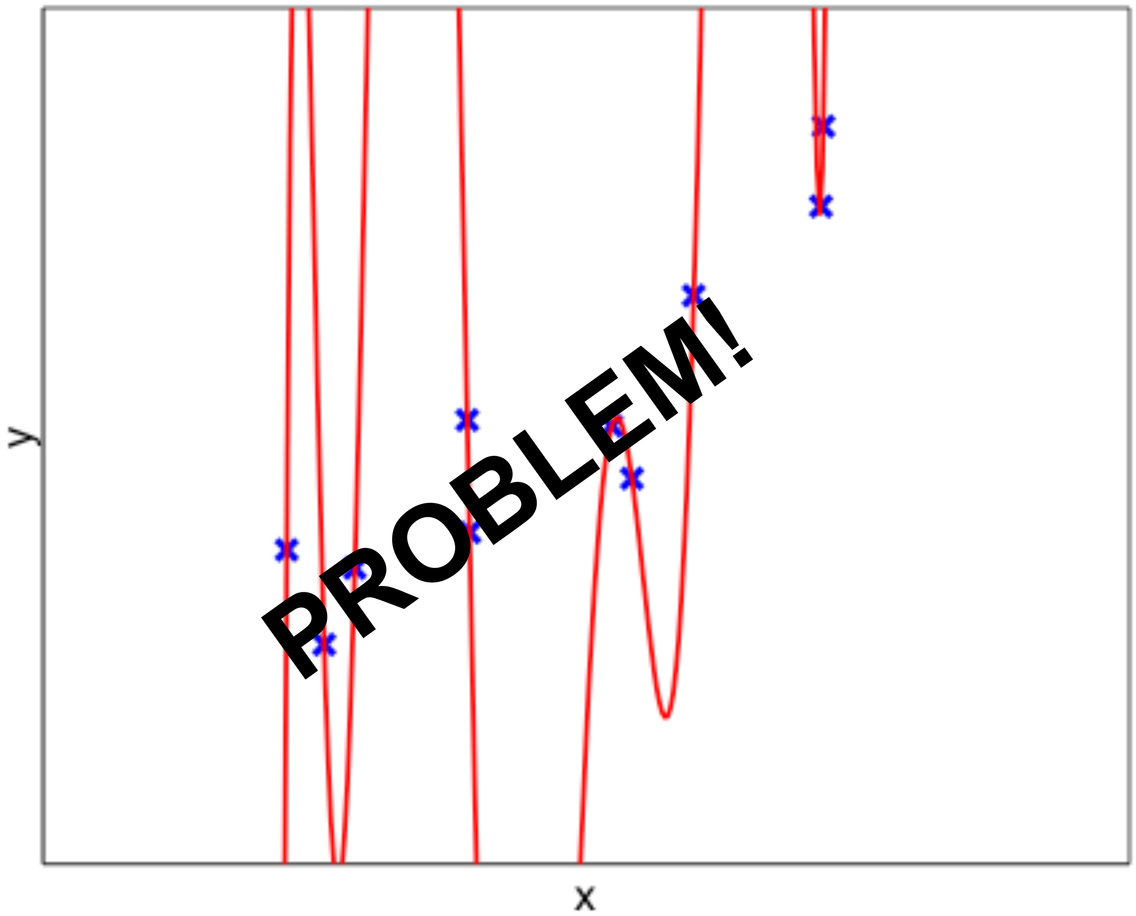
Order-7 fit



Order-8 fit



Order-9 fit



PROBLEM!

This is overfitting!

This is overfitting!

- We can find a hypothesis that explains perfectly the training data, but **does not generalize** well to new data.

- **In this example**: we have a lot of parameters (weights), so the

  hypothesis matches the data points exactly,

  but is wild everywhere else.

- A **very important** problem in machine learning.

Overfitting

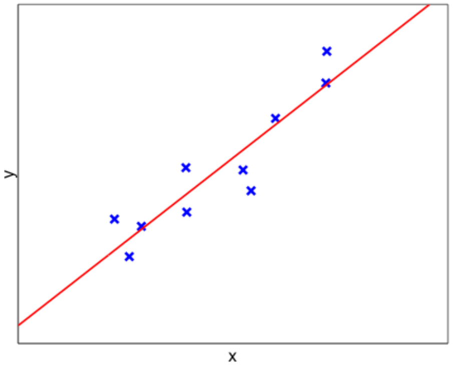
- Every hypothesis has a **true** error measured on all possible data items we could ever encounter (e.g. $f_{\mathbf{w}}(\mathbf{x}_i) - y_i$).

- Since we don't have all possible data, in order to decide what is a good hypothesis, we measure error over the training set.

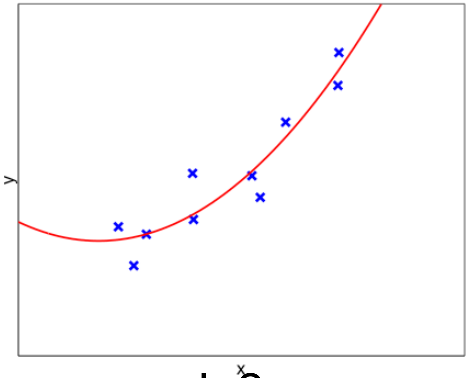
- Formally: Suppose we compare hypotheses f_1 and f_2.
  - Assume f_1 has lower error on the training set.
  - If f_2 has lower true error, then our algorithm is overfitting.

Overfitting

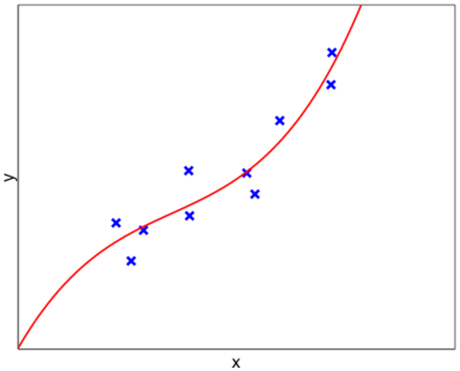
- Which hypothesis has the lowest **true** error?



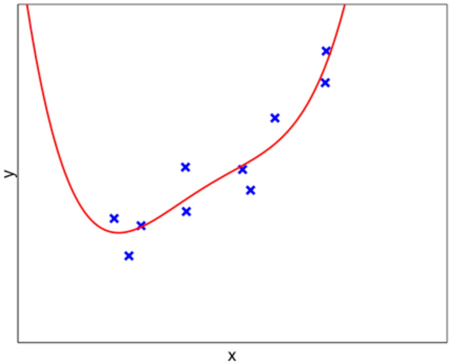
d=1



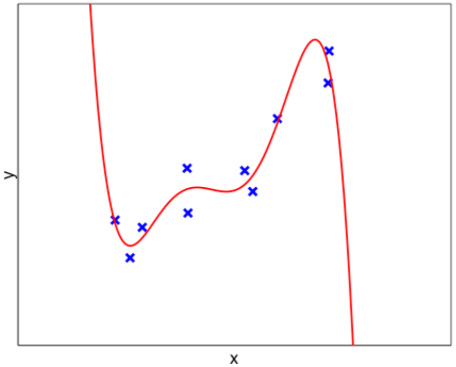
d=2



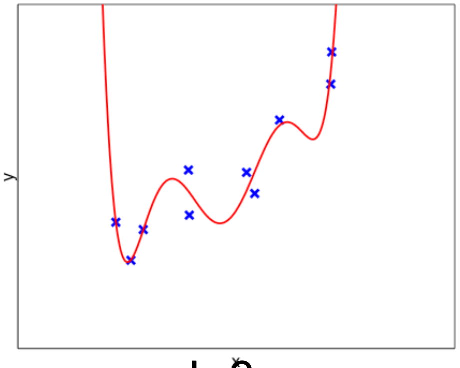
d=3



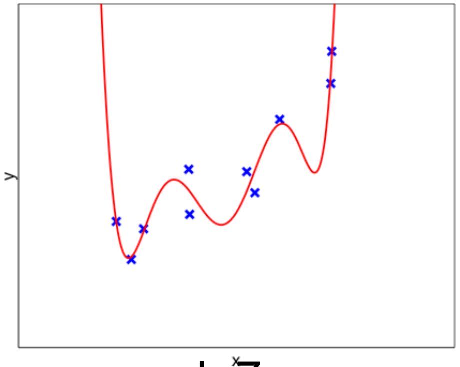
d=4



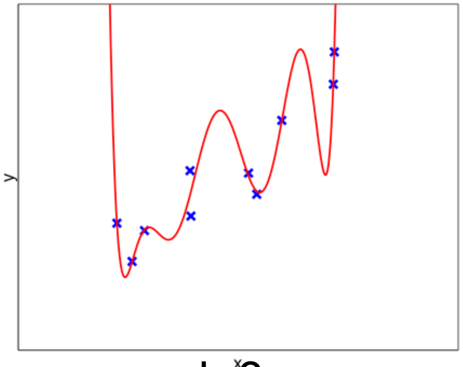
d=5



d=6



d=7



d=8

Cross-Validation

- Partition your data into a Training Set and a Validation set.
  - The proportions in each set can vary.

- Use the Training Set to find the best hypothesis in the class.

- Use the Validation Set to evaluate the true prediction error.
  - Compare across different hypothesis classes (different order polynominals.)

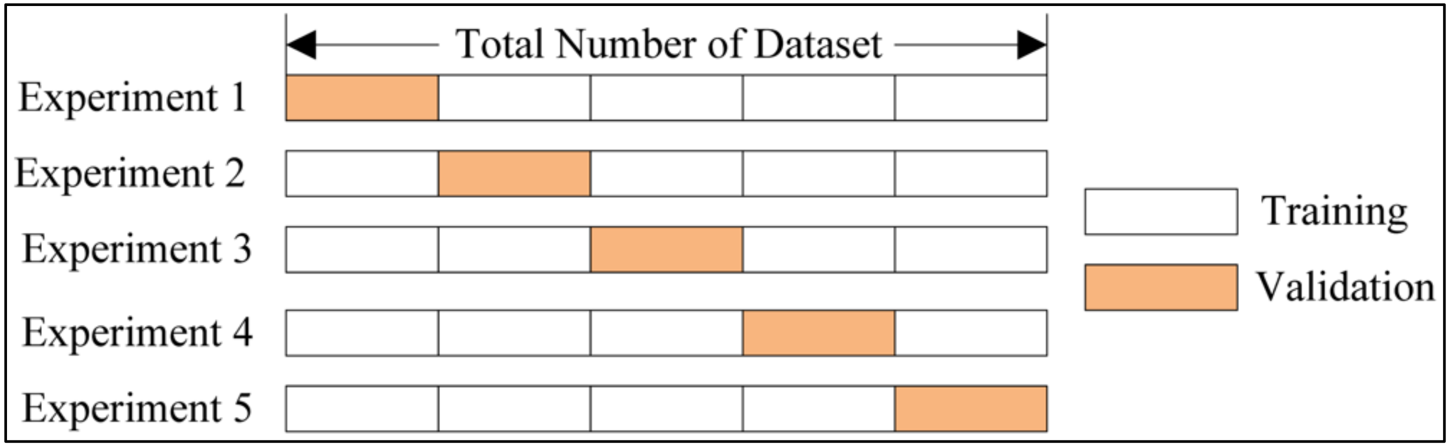
Train:

Validate:

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

k-fold Cross-Validation

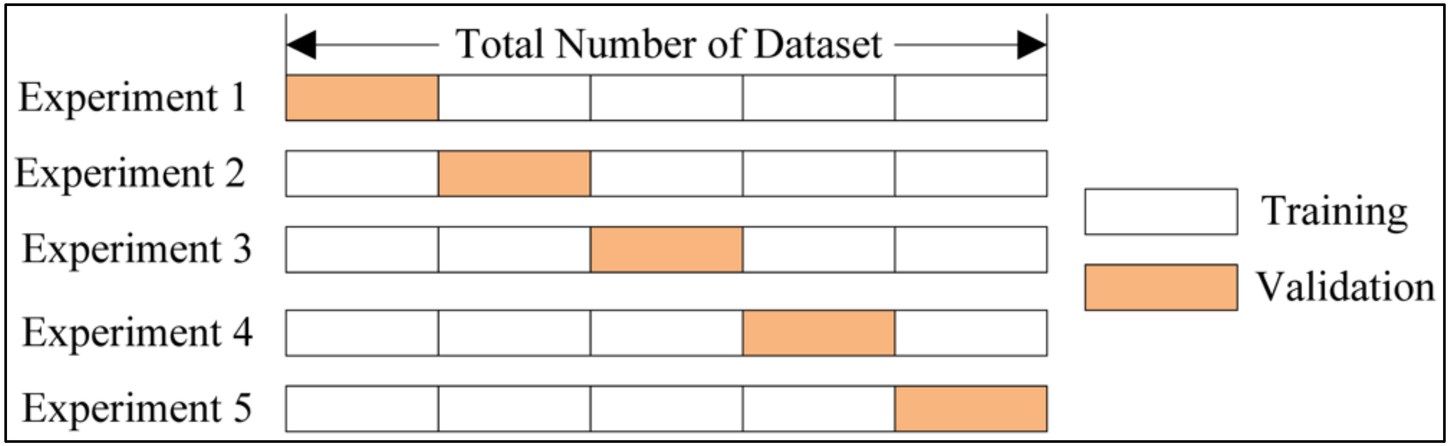
- Consider *k* partitions of the data (usually of equal size).
- Train with *k-1* subset, validate on k^{th} subset. Repeat *k* times.
- Average the prediction error over the *k* rounds/folds.



Source: http://stackoverflow.com/questions/31947183/how-to-implement-walk-forward-testing-in-sklearn

k-fold Cross-Validation

- Consider *k* partitions of the data (usually of equal size).
- Train with *k-1* subset, validate on k^{th} subset. Repeat *k* times.
- Average the prediction error over the *k* rounds/folds.



Source: http://stackoverflow.com/questions/31947183/how-to-implement-walk-forward-testing-in-sklearn

- **Computation time is increased** by factor of *k*.

Leave-one-out cross-validation

- Let $k = n$, the size of the training set

- For each order-d hypothesis class,
  - Repeat n times:
    - Set aside *one instance* $<x_i, y_i>$ from the training set.
    - Use all other data points to find $\mathbf{w}$ (optimization).
    - Measure prediction error on the held-out $<x_i, y_i>$.
  - Average the prediction error over all n subsets.

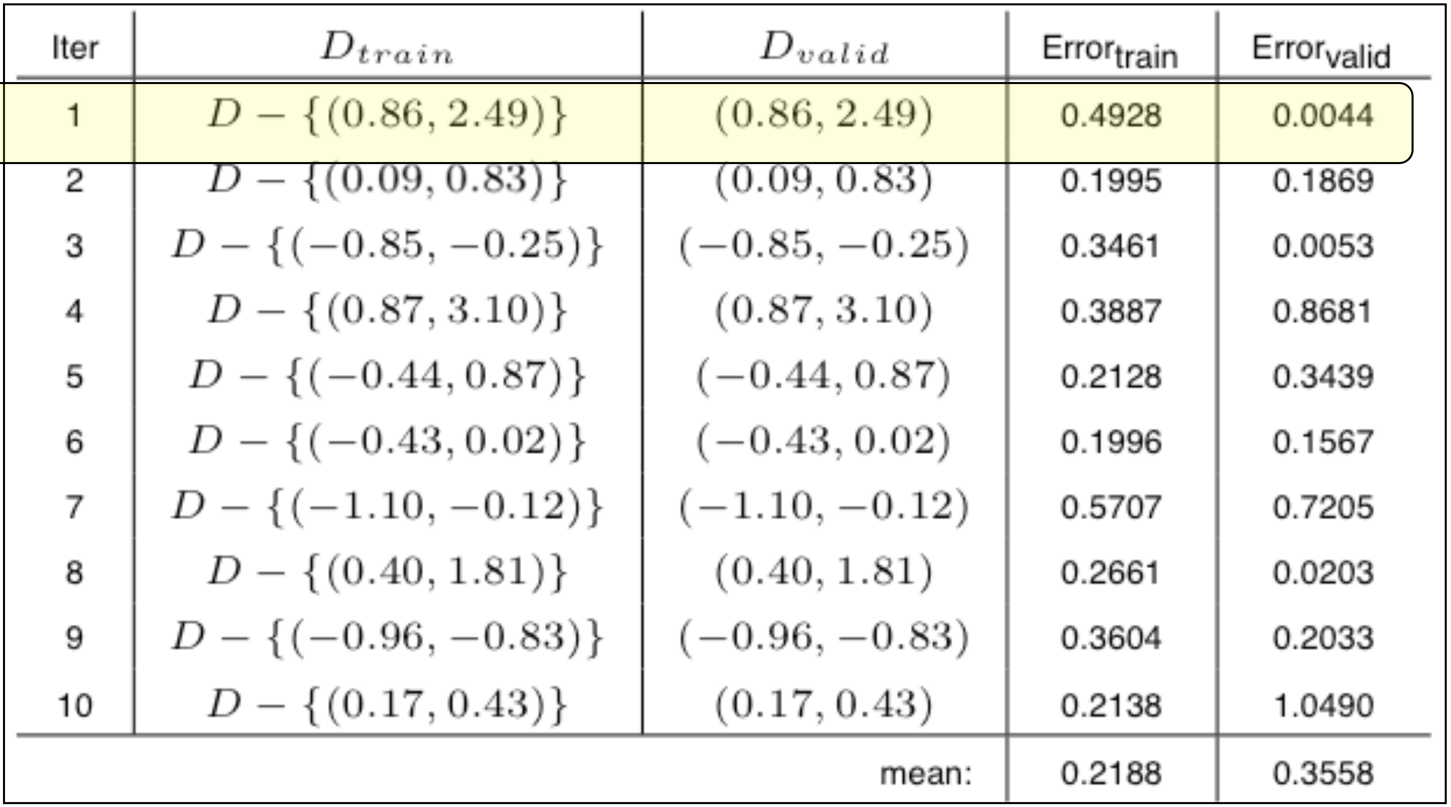
- Choose the d with lowest **estimated true prediction error**.

Estimating true error for *d*=1

Data

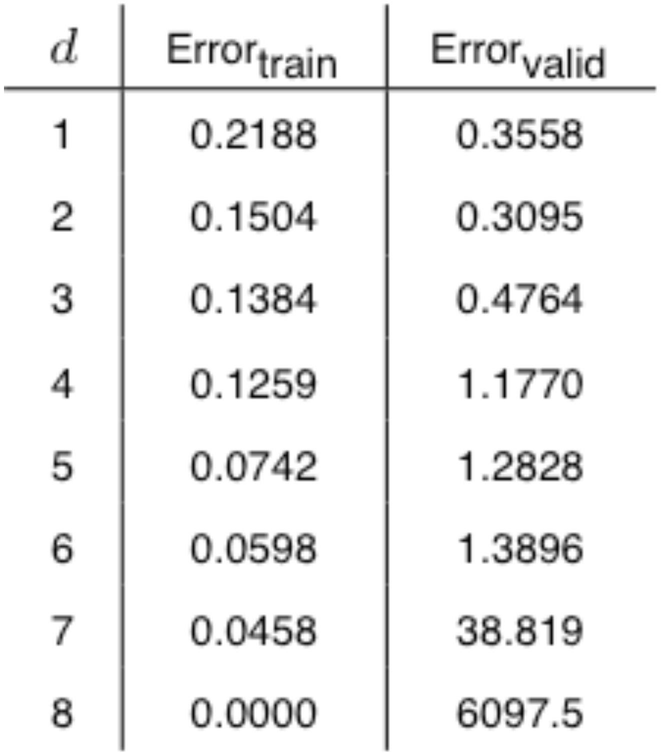
x	*y*
0.86	2.49
0.09	0.83
-0.85	-0.25
0.87	3.10
-0.44	0.87
-0.43	0.02
-1.1	-0.12
0.40	1.81
-0.96	-0.83
0.17	0.43

Cross-validation results



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

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

- Optimal choice: *d*=2. Overfitting for *d* > 2.

Evaluation

- We use cross-validation for ***model selection***.

- Available labeled data is split into two parts:
  - **Training set** is used to select a hypothesis *f* from a class of hypotheses *F* (e.g. regression of a given degree).
  - **Validation set** is used to compare the best *f* from each hypothesis class across different classes (e.g. different degree regression).
    - Must be untouched during the process of looking for *f* within a class *F*.

Evaluation

- After adapting the weights to minimize the error on the **train set,** the weights could be exploiting particularities in the **train set**:
  - have to use the **validation set** as proxy for true error
- After choosing the hypothesis class to minimize error on the **validation set,** the hypothesis class could be adapted to some particularities in the **validation set**
  - **Validation set** is no longer a good proxy for the true error!

Evaluation

- We use cross-validation for ***model selection***.

- Available labeled data is split into parts:
  - **Training set** is used to select a hypothesis *f* from a class of hypotheses *F* (e.g. regression of a given degree).
  - **Validation set** is used to compare the best *f* from each hypothesis class across different classes (e.g. different degree regression).
    - Must be untouched during the process of looking for *f* within a class *F*.
- **Test set**: Ideally, a separate set of (labeled) data is withheld to get a true estimate of the generalization error.
  - Cannot be touched during the process of selecting *F*

(Often the “validation set” is called “test set”, without distinction.)

Validation vs Train error

[From Hastie et al. textbook]

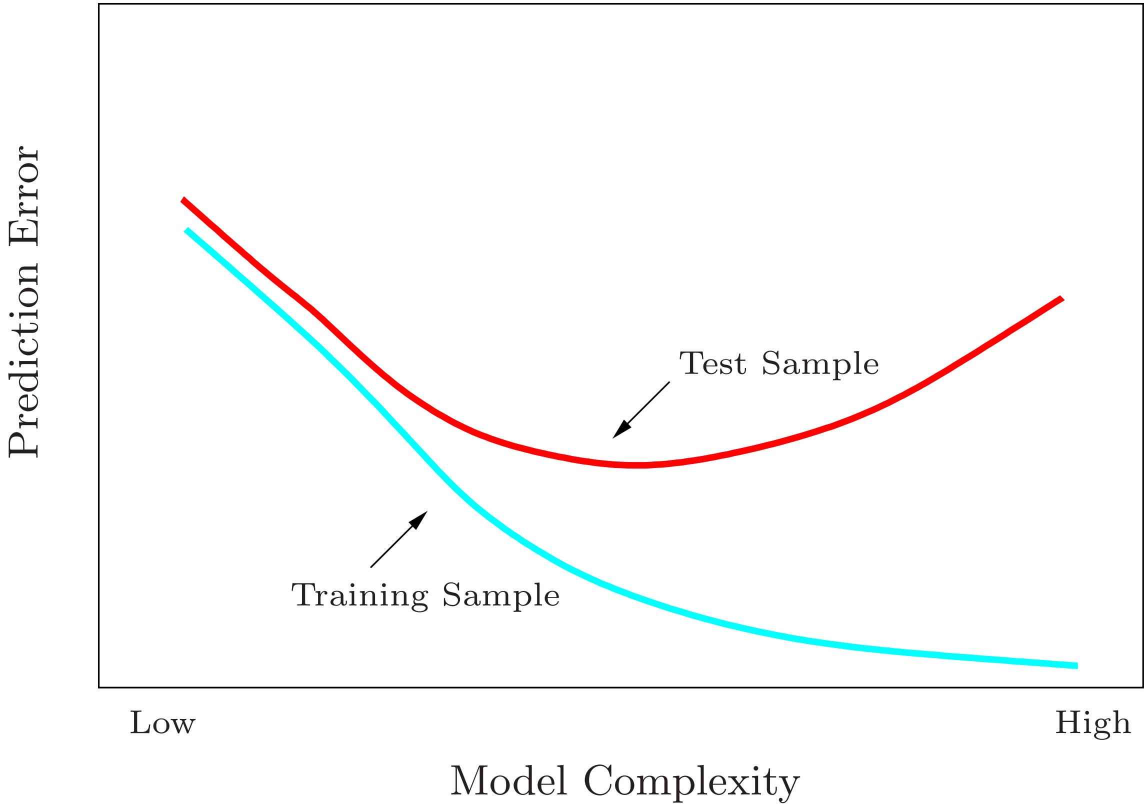
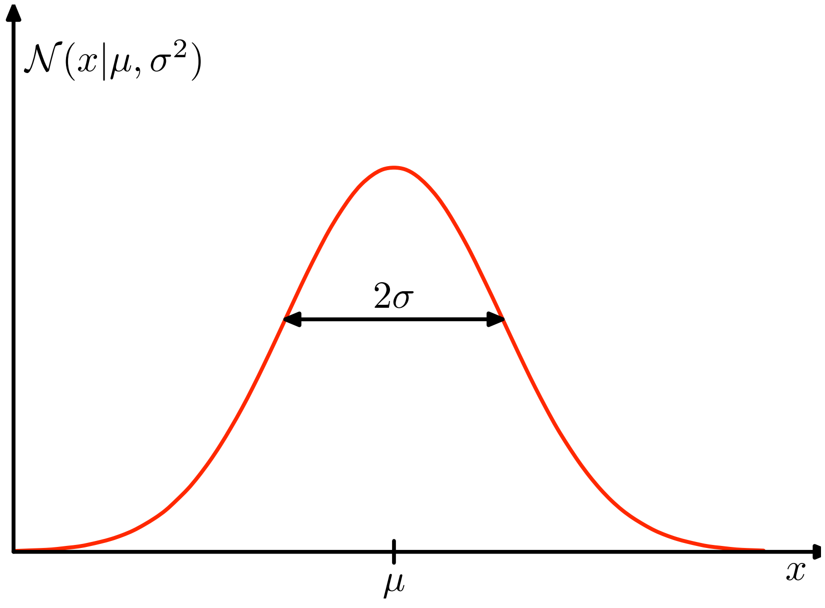


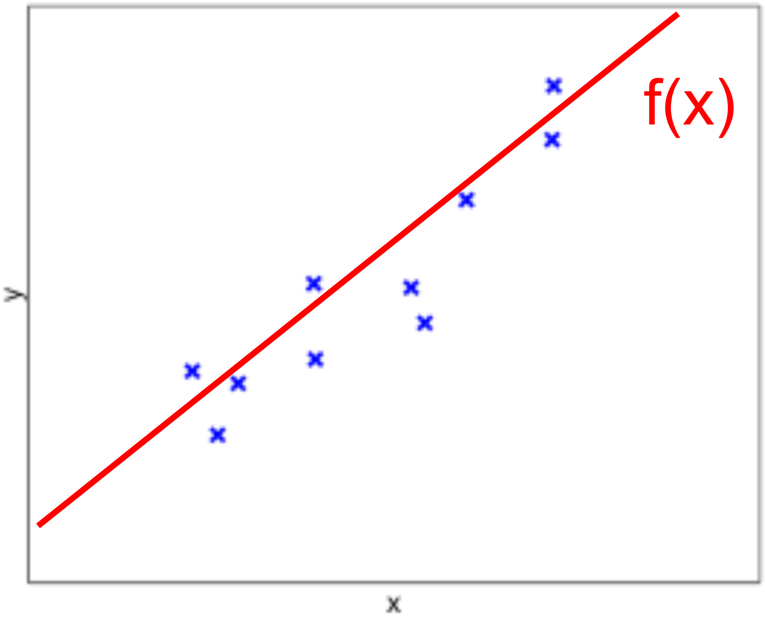
FIGURE 2.11. *Test and training error as a function of model complexity.*

Understanding the error

Given set of examples <X, Y>. Assume that $y = f(x) + \epsilon$,

where ϵ is Gaussian noise with zero mean and std deviation σ.





Understanding the error

- Consider standard linear regression solution:

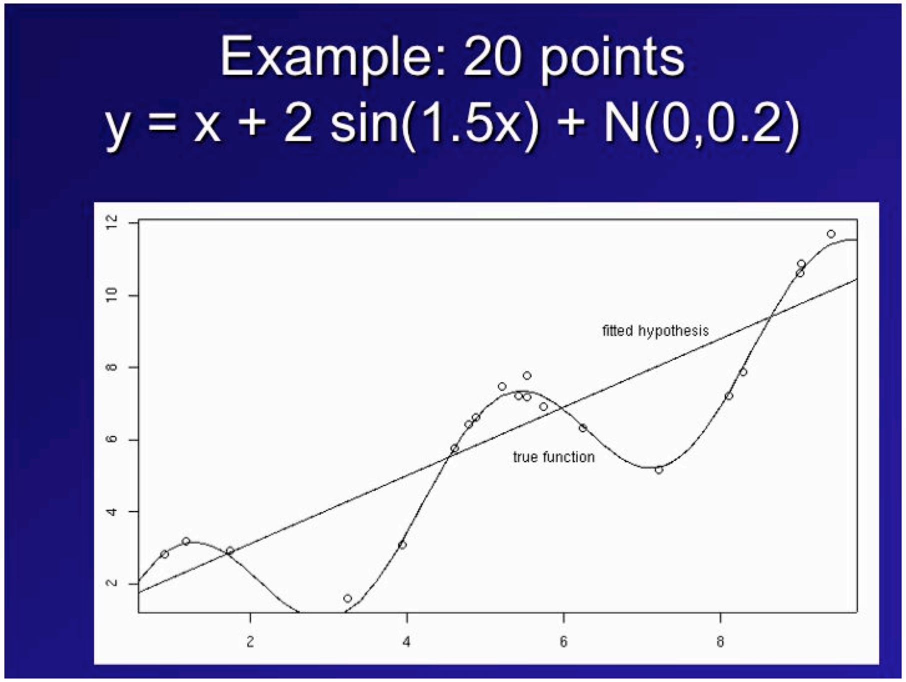
$$Err(\mathbf{w}) = \sum_{i=1:n} (y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

- If we consider only the class of **linear** hypotheses, we have systematic prediction error, called **bias**, whenever the data is generated by a non-linear function.

- Depending on what dataset we observed, we may get different solutions. Thus we can also have error due to this **variance**.
  - This occurs even if data is generated from class of linear functions.

An example (from Tom Dietterich)

- The circles are data points. X is drawn uniformly randomly. Y is generated by the function *$y = 2\sin(0.5x) + \epsilon$*.

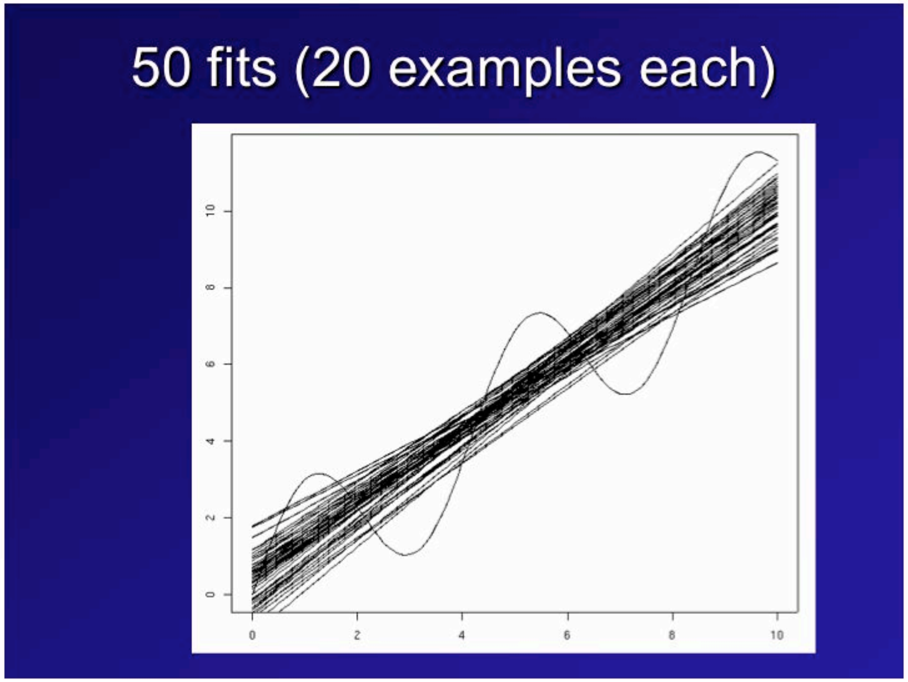


Example: 20 points

$y = x + 2 \sin(1.5x) + N(0,0.2)$

An example (from Tom Dietterich)

- With different sets of 20 points, we get different lines.



50 fits (20 examples each)

Validation vs Train error

[From Hastie et al. textbook]

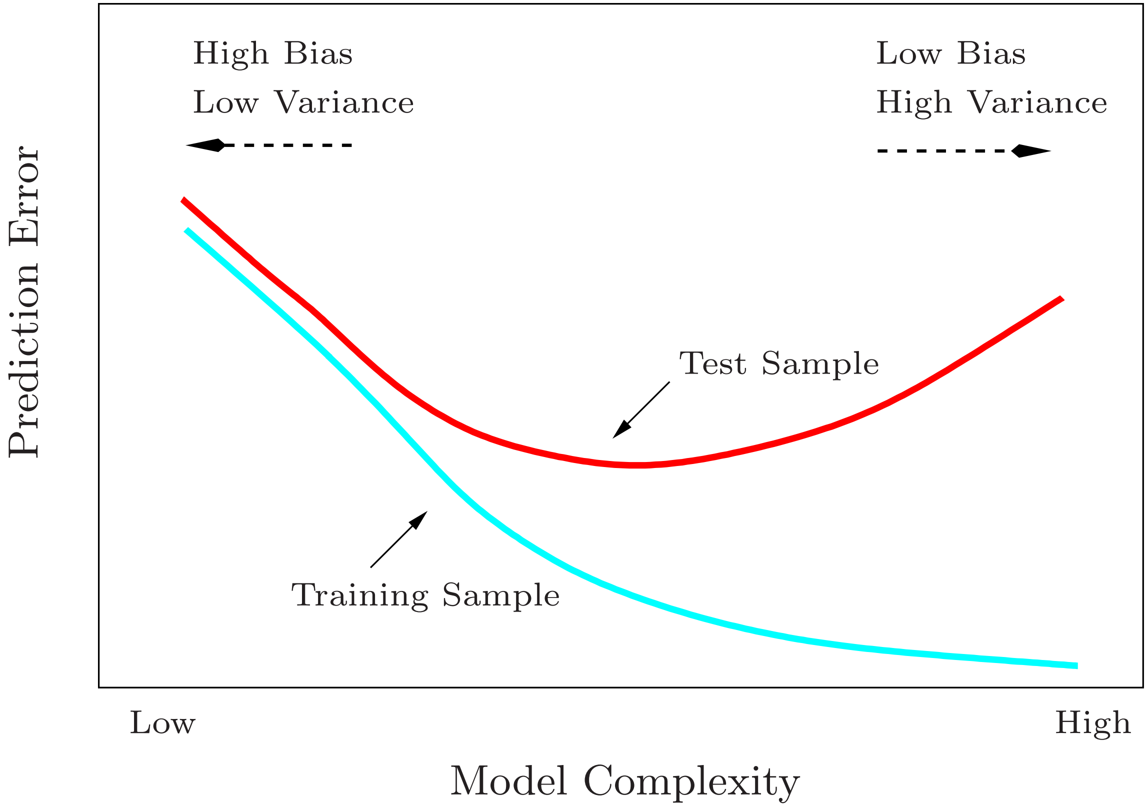


FIGURE 2.11. *Test and training error as a function of model complexity.*

Gauss-Markov Theorem

- Main result:

The least-squares estimates of the parameters ***w*** have the **smallest variance** among all linear **unbiased** estimates.

Gauss-Markov Theorem

- Main result:

  The least-squares estimates of the parameters $\mathbf{w}$ have the **smallest variance** among all linear **unbiased** estimates.

- Understanding the statement:
  - Real parameters are denoted: $\mathbf{w}$
  - Estimate of the parameters is denoted: $\hat{\mathbf{w}}$
  - Error of the estimator: $Err(\hat{\mathbf{w}}) = E(\hat{\mathbf{w}}-\mathbf{w})^2 = Var(\hat{\mathbf{w}}) + (E(\hat{\mathbf{w}}-\mathbf{w}))^2$
  - Unbiased estimator means: $E(\hat{\mathbf{w}}-\mathbf{w})=0$
  - There may exist an estimator that has lower error, but some bias.

Bias vs Variance

- Gauss-Markov Theorem says:

  The least-squares estimates of the parameters ***w*** have the **smallest variance** among all linear **unbiased** estimates.

- **Insight**: Find lower variance solution, at the expense of some bias.

Bias vs Variance

- Gauss-Markov Theorem says:

  The least-squares estimates of the parameters ***w*** have the **smallest variance** among all linear **unbiased** estimates.

- **Insight**: Find lower variance solution, at the expense of some bias.

- E.g. fix low-relevance weights to 0

Recall our prostate cancer example

- The Z-score measures the effect of dropping that feature from the linear regression.

$z_j = \hat{w}_j / sqrt(\sigma^2 v_j)$

where $\hat{w}_j$ is the estimated weight of the j^{th} feature, and v_j is the j^{th} diagonal element of $(X^TX)^{-1}$

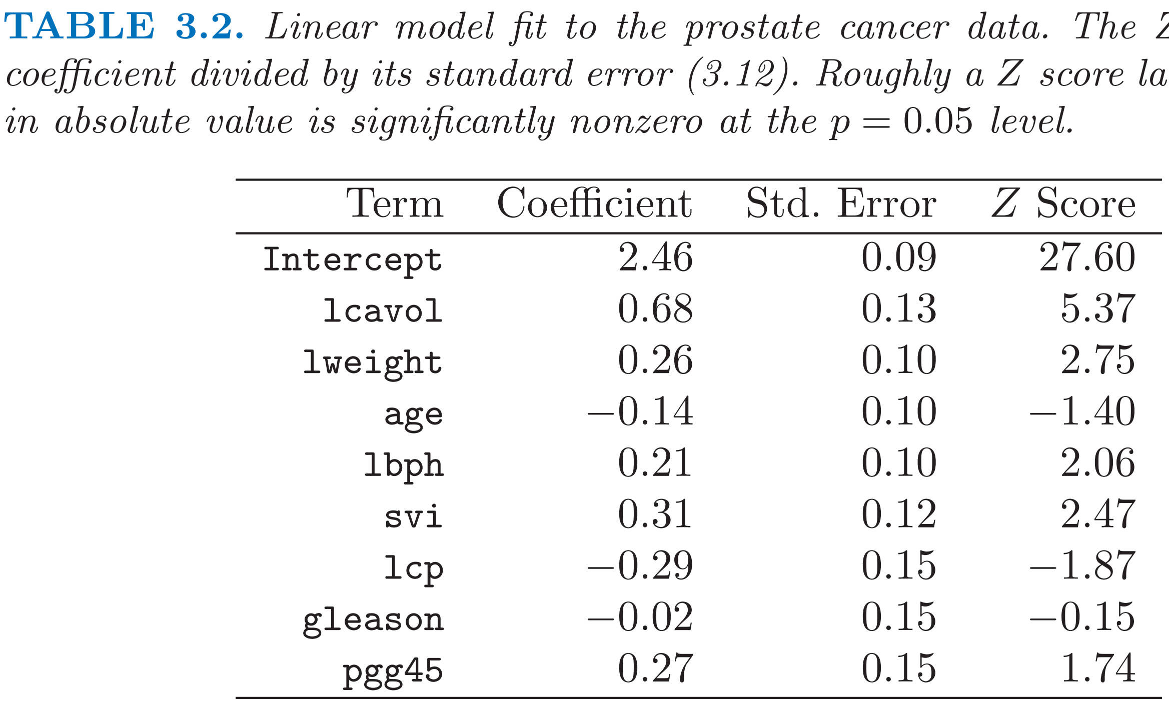


TABLE 3.2. *Linear model fit to the prostate cancer data. The Z score is the coefficient divided by its standard error (3.12). Roughly a Z score larger than two in absolute value is significantly nonzero at the $p = 0.05$ level.*

Term	Coefficient	Std. Error	Z Score
`Intercept`	2.46	0.09	27.60
`lcavol`	0.68	0.13	5.37
`lweight`	0.26	0.10	2.75
`age`	−0.14	0.10	−1.40
`lbph`	0.21	0.10	2.06
`svi`	0.31	0.12	2.47
`lcp`	−0.29	0.15	−1.87
`gleason`	−0.02	0.15	−0.15
`pgg45`	0.27	0.15	1.74

[From Hastie et al. textbook]

Subset selection

- **Idea**: Keep only a small set of features with non-zero weights.
- **Goal**: Find lower variance solution, at the expense of some bias.
- There are many different methods for choosing subsets.

  (More on this later…)
- Least-squares regression can be used to estimate the weights of the selected features.
- Bias as true model might rely on the discarded features!

Bias vs Variance

- Find lower variance solution, at the expense of some bias.

- Force some weights to 0
- E.g. Include **penalty** for model complexity in error to reduce overfitting.

$$Err(w) = \sum_{i=1:n} (y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\ |model\_size|$$

λ is a hyper-parameter that controls penalty size.

Ridge regression (aka L2-regularization)

- Constrains the weights by imposing a penalty on their size:

$$\hat{\mathbf{w}}^{ridge} = argmin_{\mathbf{w}} \{ \sum_{i=1:n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda \boxed{\sum_{j=0:m} w_j^2} \}$$

where λ can be selected manually, or by cross-validation.

Ridge regression (aka L2-regularization)

- Constrains the weights by imposing a penalty on their size:

  $$\hat{\mathbf{w}}^{ridge} = argmin_{\mathbf{w}} \{ \sum_{i=1:n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\sum_{j=0:m}w_j^2 \}$$

  where λ can be selected manually, or by cross-validation.

- Do a little algebra to get the solution: $\hat{\mathbf{w}}^{ridge} = (X^TX+\lambda I)^{-1}X^TY$

Ridge regression (aka L2-regularization)

- Re-write in matrix notation: $f_{\mathbf{w}}(X) = X\mathbf{w}$

  $Err(\mathbf{w}) = (Y - X\mathbf{w})^T(Y - X\mathbf{w}) + \lambda \mathbf{w}^T\mathbf{w}$

- To minimize, take the derivative w.r.t. $\mathbf{w}$:

  $\partial Err(\mathbf{w})/\partial \mathbf{w} = -2\, X^T\, (Y - X\mathbf{w}) - 2\lambda \mathbf{w} = \mathbf{0}$

- Try a little algebra:

Ridge regression (aka L2-regularization)

- Re-write in matrix notation: $f_{\mathbf{w}}(X) = X\mathbf{w}$

  $Err(\mathbf{w}) = (Y - X\mathbf{w})^T(Y - X\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w}$

- To minimize, take the derivative w.r.t. $\mathbf{w}$:

  $\partial Err(\mathbf{w})/\partial\mathbf{w} = -2\,X^{\mathbf{T}}\,(Y - X\mathbf{w}) - 2\lambda\mathbf{w} = \mathbf{0}$

- Try a little algebra: $X^T\,Y = (X^T\,X + \mathbf{I}\lambda)\,\mathbf{w}$

  $\hat{\mathbf{w}} = (X^TX + \mathbf{I}\lambda)^{-1}\,X^T\,Y$

Ridge regression (aka L2-regularization)

- Constrains the weights by imposing a penalty on their size:

  $$\hat{\mathbf{w}}^{ridge} = argmin_{\mathbf{w}} \{ \sum_{i=1:n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \sum_{j=0:m} w_j^2 \}$$

  where λ can be selected manually, or by cross-validation.

- Do a little algebra to get the solution: $\hat{\mathbf{w}}^{ridge} = (X^T X + \lambda I)^{-1} X^T Y$
  - The ridge solution is not equivariant under scaling of the data, so typically need to normalize the inputs first.
  - Ridge gives a smooth solution, effectively shrinking the weights, but drives few weights to 0.

Lasso regression (aka L1-regularization)

- Constrains the weights by penalizing the absolute value of their size:

$$\hat{\mathbf{w}}^{lasso} = argmin_W \{ \sum_{i=1:n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda \boxed{\sum_{j=1:m}|w_j|} \}$$

Lasso regression (aka L1-regularization)

- Constrains the weights by penalizing the absolute value of their size:

$$\hat{\mathbf{w}}^{lasso} = argmin_W \{\ \sum_{i=1:n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\sum_{j=1:m}|w_j|\ \}$$

- Now there is no closed-form solution. Need to solve a quadratic programming problem instead.

Lasso regression (aka L1-regularization)

- Constrains the weights by penalizing the absolute value of their size:

$$\hat{\mathbf{w}}^{lasso} = argmin_W \{ \; \sum_{i=1:n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\sum_{j=1:m}|w_j| \; \}$$

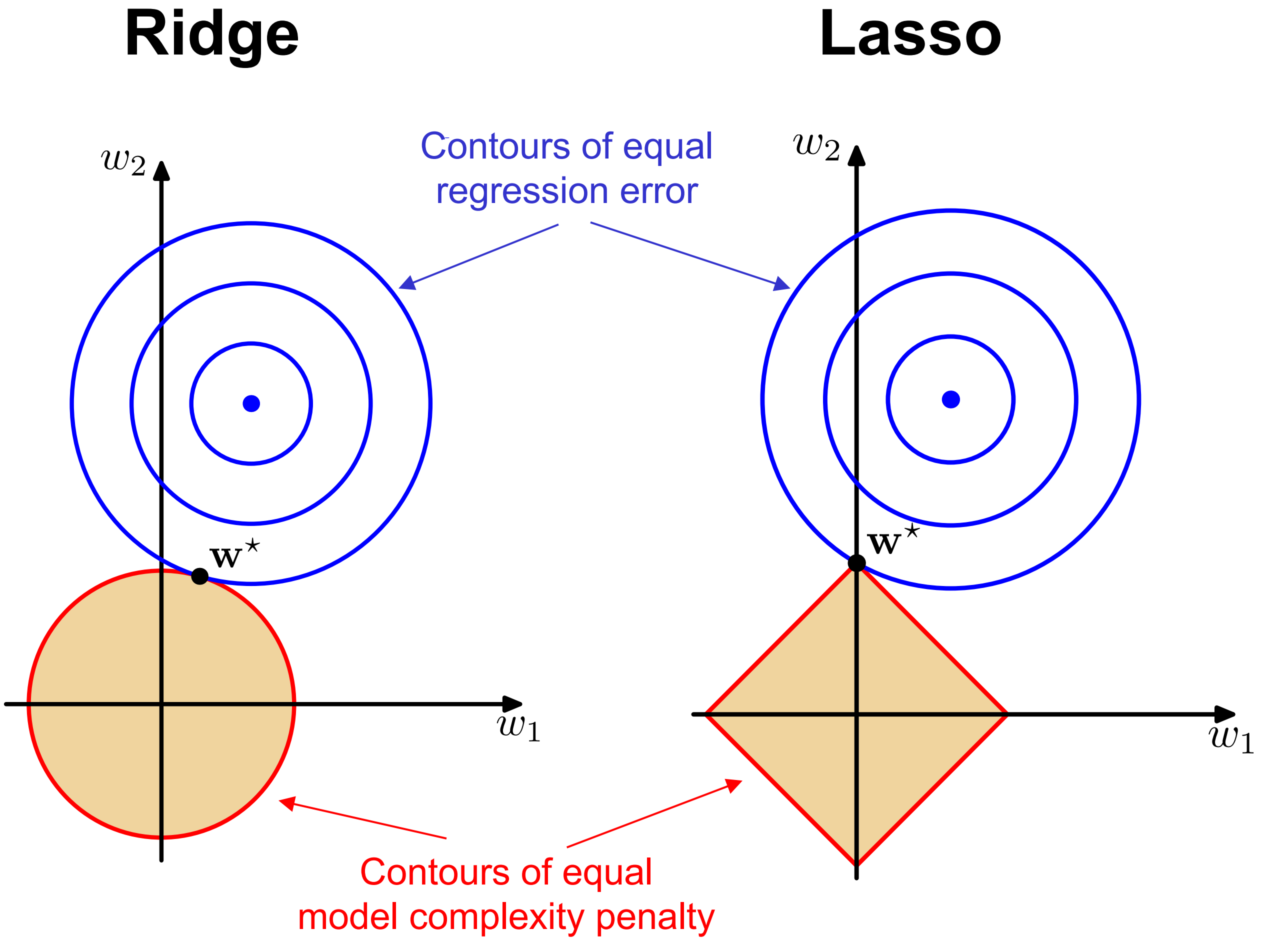
- Now there is no closed-form solution. Need to solve a quadratic programming problem instead.
  - More computationally expensive than Ridge regression.
  - Effectively sets the weights of less relevant input features to zero.

Comparing Ridge and Lasso

	Ridge	**Lasso**
Complexity:	$\sum_{j=0:m} w_j^2$	$\lambda\sum_{j=1:m}\lvert w_j\rvert$

- Note that for lasso, reducing any weight by, say, 0.1 reduces the complexity by the same amount
  - So, we’d prefer reducing less relevant features
- In ridge regression, reducing high weights reduces complexity more than reducing a low weight
  - So, trade-off between reducing less relevant features and larger weights. Tend to not reduce weights that are already small

Comparing Ridge and Lasso



Ridge

Lasso

Contours of equal regression error

Contours of equal model complexity penalty

A quick look at evaluation functions

- We call $L(Y,f_{\mathbf{w}}(x))$ the loss function.
  - Least-square / Mean squared-error (MSE) loss:

$$L(Y, f_{\mathbf{w}}(X)) = \sum_{i=1:n} (y_i - \mathbf{w}^T x_i)^2$$

- Other loss functions?

A quick look at evaluation functions

- We call $L(Y,f_{\mathbf{w}}(x))$ the loss function.
  - Least-square / Mean squared-error (MSE) loss:

    $$L(Y, f_{\mathbf{w}}(X)) = \sum_{i=1:n} (y_i - \mathbf{w}^T x_i)^2$$

- Other loss functions?
  - Absolute error loss: $L(Y, f_{\mathbf{w}}(X)) = \sum_{i=1:n} | y_i – \mathbf{w}^T x_i |$
  - 0-1 loss (for classification): $L(Y, f_{\mathbf{w}}(X)) = \sum_{i=1:n} I (y_i \neq f_{\mathbf{w}}(x_i))$

A quick look at evaluation functions

- We call $L(Y,f_{\mathbf{w}}(x))$ the loss function.
  - Least-square / Mean squared-error (MSE) loss:

$$L(Y, f_{\mathbf{w}}(X)) = \sum_{i=1:n} (y_i - \mathbf{w}^T x_i)^2$$

- Other loss functions?
  - Absolute error loss: $L(Y, f_{\mathbf{w}}(X)) = \sum_{i=1:n} | y_i - \mathbf{w}^T x_i |$
  - 0-1 loss (for classification): $L(Y, f_{\mathbf{w}}(X)) = \sum_{i=1:n} I (y_i \neq f_{\mathbf{w}}(x_i))$

- Different loss functions make **different assumptions**.
  - Squared error loss assumes the data can be approximated by a global linear model with Gaussian noise.
  - Loss function **independent** of complexity penalty (l1 or l2)